

METHOD AND APPARATUS FOR CONTROLLING COMMUNICATIONS

From a first, general aspect, the present invention relates to a method and apparatus for preparing and processing information to be sent or received via a network. A network in this instance may be implemented as data carried either over communications lines and/or stored on smart cards (or other data carriers) and physically transported.

From a second, more specific aspect, the present invention relates to a method and apparatus for controlling remote payment transactions, particularly, but not exclusively, for controlling remote payment transactions where a persons account is credited and/or debited from a remote location in exchange for goods/services cash or credit, or where account information is accessed remotely to enable approval of a transaction.

Devices for carrying out remote payment transactions are well-known. These "payment terminals" include EFTPOS, credit card payment terminals, etc.

The most common function of payment terminals is to remotely access a persons account information and either carry out a transaction, such as crediting or debiting the account, or, particularly in the case of credit card payment terminals, to check the users account to ensure that there are sufficient funds to cover a transaction. Note that although credit card terminals do not necessarily remotely credit or debit the users account (the credit/debit transaction usually being carried out by a separate paper bill trail) and merely provide the information that the users account is sufficient to cover the transaction, such payment terminals still fall within the ambit of the present invention and the term "transaction" as used herein includes the operation of remotely checking the users account to "ok" a transaction.

A payment terminal may, for example, provide for the following basic operations:

(1) Input of information which is required to enable access to a customers account. The information is most often read from a magnetic stripe on a credit card or bank card or the like, or a smart card. In addition to reading details from a card a personal identification number (PIN) or the like code may also be required.

(2) Obtain access to the customers account. This is usually done by remote communication with a processing device holding the person's account data, usually on bank premises and remote from the payment terminal. Usually, information on the customers account input to the payment terminal will need to be transmitted for verification and to enable access to the account. Also a money amount will usually need to be input to the payment terminal and transmitted over the communications line. At least some and perhaps all of the transmitted data may be encrypted for security purposes and the payment terminal is therefore, in such a case, required to have means (3) providing encryption.

(4) The payment terminal may need to be able to receive communications over the remote line from the processor accessing the customers account, ie. to provide an "answer" to the payment device regarding the user transaction. The answer may include information that an account debit/credit has taken place (eg. EFTPOS) or merely an approval that the customer has enough money in his account to enable a transaction (some credit card payment terminals). Again, this transmitted information may be encrypted and, if so, will require translation (5) in the payment terminal.

(6) To provide an indication that the transaction request is approved or that a transaction has occurred, by display or printer, for example. Displays may also be required to prompt an operator or customer to input information, e.g., input your PIN "Input Amount".

There are many different brands of payment terminal, utilising many different software and hardware arrangements. This gives rise to a number of problems.

Any account acquirer (eg. bank) will generally have
5 their own operating requirements as to how remote payment trans-actions will be handled. The account acquirer may purchase a series of payment terminals which have been configured by a manufacturer to the acquirer's requirements. These payment terminals will then be
10 licensed or rented or more often supplied at no charge to merchants (e.g., retail stores, garages, restaurants). Multiple account acquirers may require access to their customers accounts via a single payment terminal. That is, one particular merchant may operate payment terminals which
15 provide access to customers accounts at other account acquirers (e.g., other banks). Because of different requirements of different acquirers for handling of remote payment transactions, the payment terminal must be arranged to operate to satisfy the different requirements.

20 The terminal owner (often a principle acquirer) will have the terminal appropriately arranged and programmed by the terminal manufacturer to satisfy the requirements of all account acquirers utilising the terminal. Payment terminals may need to contain several programs and select
25 the appropriate program depending on the card to be processed or on an operator selection.

It is often the case that the terminal owner may need to have the operation of the payment device amended to, for example, enable it to operate for an additional account
30 acquirer, or to satisfy changed requirements for a particular account acquirer. Because of the different hardware/software architectures available, any operational alterations generally the require the input of the terminal supplier or manufacturer. The supplier/manufacturer will be
35 required to reprogram the terminal or amend the hardware in order to carry out the alterations and they will usually be

the only person who has the appropriate knowledge. The terminal owner is thus tied to the particular supplier/manufacturer of the particular brand of payment terminal.

5 It is often the case that, the terminal owner may over time obtain different brands from different manufacturers and for operational alterations may need to return the particular brand to each separate manufacturer. Over time, manufacturers may go out of business, in which case the
10 payment terminals made by that particular manufacturer may be unsupported and any alteration may be difficult to achieve, or at least will require the input of a skilled person having detailed knowledge of the programming and/or hardware of the redundant manufacturer's devices.

15 Being tied to a particular manufacturer for a particular brand therefore causes cost, time and trouble when any operational alterations are required. There is therefore a reluctance to carry out operational alterations, which sometimes means that requirements of
20 various account acquirers are not fully satisfied. When an operational alteration does have to be carried out, it is costly. If a manufacturer goes out of business, the terminal owner may be left with nobody to alter the operation of his payment terminals, or indeed maintain the
25 payment terminals. The present system is costly and inflexible.

 A payment terminal device usually includes a microprocessor and a number of peripheral units (e.g., card reader, display, printer, communications interface, __etc)
30 controlled by the processor. A payment terminal device usually comprises hardware, an operating system or a BIOS and is ready to accept an application for that arrangement. Or the device may be supplied with an interpreter to accept the applications.

35 To alter the operation of payment terminals, a new application must be created. This can be time consuming,

costly and as the programming will be different for different types of devices, which may have different hardware arrangements as well, and must be carried out separately for each different type of device (i.e.,
5 different reprogramming operations must be carried out for different devices even where the same operational alterations may be required).

The programming alterations are not "portable" between different types of devices.

10 The most time critical aspects of operation of a remote payment terminal involve the building up and breaking down of "messages" and the formulation and operation of communications. By "messages" is meant, for example, information data which is required to be input to
15 the device or communicated or displayed in order to enable carrying out of a remote payment transaction, and includes information to be communicated to the bank, e.g., customers card number, customers PIN, amount of transaction, etc; displayed information such as "Please Input Amount";
20 information to be read from a customers magnetic stripe card or smart card and manipulated by the device e.g., card number, expiry date, etc. The operation of payment terminals is greatly concerned with the collection, rearrangement and communication of this message data to
25 enable a remote payment transaction.

In conventional devices, each time a message is constructed or deconstructed, the operation of the machine will be handled by the application program. To change operation of the machine, the application must be changed.
30 This is laborious, and gives rise to problems, as discussed above.

The technique of creating a virtual processor (or in this case microprocessor) is well known and referred to as an interpreter. This allows programs to operate
35 independent of processor. With the newer technique of also creating virtual peripherals then the whole is

referred to as a "virtual machine".

A virtual machine is computer programmed to emulate a hypothetical computer. Different incompatible computers may be programmed to emulate the same hypothetical computer.

5 Any computer programmed to emulate the hypothetical computer will thus be capable of executing programs for the virtual computer. This creates a complete portable environment for program operations.

10 A problem with virtual machines is emulation is slower than normal program execution. For some applications this performance penalty is a significant problem.

15 The above problems and disadvantages which have been discussed specifically in relation to devices configured to process payment transactions also would apply to devices configured to prepare and process any information to be sent or received via a network, not restricted to payment transaction information.

20 From a first aspect the present invention provides a communications device which is arranged to process messages for communications, comprising a virtual machine means which includes a virtual function processor and function processor instructions for controlling operation of the device, and a virtual message processor which is arranged to be called by the function processor and which is
25 arranged to carry out the task of assembling, disassembling and comparing messages, whereby when a message is required to be handled by the communications device the message processor is called to carry out the message handling task.

30 "Communications" includes transport of data via a data carrier such as a smart card.

By messages we mean a sequence of data comprising usually a plurality of information fields to be communicated.

35 The message processor means is preferably translated into the native code of the microprocessor in each hardware device on which the virtual machine is to be

implemented. The message processor instructions are preferably virtual instructions to be expressed only in the language defined by the message processor means- and thus never requiring translation to any real hardware processor.

5 The message processor means in at least a preferred embodiment provides two specific advantages over conventional arrangements

1) Faster Operation. The processor (executing as native code) operates at full microprocessor speed
10 overcoming the problem of slow emulation speed for message related functions.

2) Faster, simpler programming. The instructions for the message processor preferably consist of actual message "descriptions". The programmer need only describe
15 the message content, all data conversion, manipulation and processing is automatically performed based on the message description. This is a more intuitive and compartmentalised approach which preferably leads to faster programming with less errors.

20 The protocol processor means is preferably a program module the specific function of which is to control and select the sequence of message processor operations in relation to messages received and transmitted.

25 The protocol processor means is preferably translated into the native code of the microprocessor in each hardware device on which the virtual machine is to be implemented. The protocol processor instructions are virtual instructions expressed only in the language defined by the protocol processor means and thus never requiring
30 translation to any real hardware processor. The protocol processor means provides two specific advantages over conventional arrangements:

1) Faster Operation. The processor (executing as native code) preferably operates at full microprocessor
35 speed overcoming the problem of slow emulation speed for

protocol related functions.

2) Faster, simpler programming. The instructions for the protocol engine preferably consist of an actual diagram of the message flow. To change message flow or sequence, the programmer can modify an intuitive diagram, all multi-processing and other complications are handled automatically. This more intuitive and compartmentalised approach leads to faster programming with less errors.

In a preferred embodiment, therefore, a device in accordance with the present invention includes a virtual machine including virtual processors which are specifically arranged to control message construction, deconstruction, comparison and to control the communication of information, both for reception from a network and transmission to a network. These operations can therefore be carried out at speed, overcoming the problems with known virtual machines and interpreters, which tend to operate slower than conventionally programmed devices. The virtual machine therefore lends itself particularly to applications relating to communications, such as payment terminal devices and other devices in which message processing and communication comprise a significant proportion of the operation of the device. In payment terminals, for example, a payment terminal including a virtual machine having the message processor means and protocol processor means can operate satisfactorily speedwise. The virtual machine can be implemented on any hardware, BIOS/OS arrangement and therefore facilitates portability of programs.

Implementation of such a virtual machine on payment terminal devices of different brands enables operation of the payment terminal devices or brands to be altered merely by altering application commands generic to all brands. Each brand is seen by the application as the same virtual machine.

The virtual machine preferably also includes a function processor means arranged to control overall virtual machine action in response to operator or other external events, and also preferably includes function processor instructions which are arranged to provide
5 directions for operation of the function processor means.

The function processor means is preferably a program module the specific function of which is to control and select general operations of the device not specially controlled by the message and protocol processor means.
10

The function processor means is preferably translated into the native code of the microprocessor in each hardware device on which the virtual machine is to be implemented. The function processor instructions are preferably virtual instructions to be expressed only in the language defined by the function processor means- and thus never requiring translation to any real hardware processor.
15

In the preferred embodiment, the "application" will therefore comprise instructions for the message, protocol and function processor means. The instructions for the function processor means may include such prior art modules as a function event scheduler and function selector.
20

Although the present invention is particularly applicable to application in payment terminals, it is not limited to such applications. The invention can be applied in any device where advantages are likely to be achieved for the arrangement and control of communications.
25

With the advent of the Internet and other extensive communications networks, it is believed that the operation of computers, such as PC's, will become more and more oriented towards acting as "servers" and/or "browsers". In other words, a major function of PC's connected to a network will be to operate either as a server, providing information and/or programs to the network for access by
30 other parties, or as a "browser" for obtaining
35

information/programs available on the network and operating on them. It is likely, in fact, that PC's will be asked to operate as both a server and a browser. This operation will not merely be restricted to the Internet, but for any
5 network, even Local Area Net-works.

The applicant also believes that many other classes of devices may be connected to a network. For example in the future a home video cassette recording machine could be connected to the Internet (along with other devices)
10 allowing remote programming from a browser device. An example of the use of this would be a worker upon learning of a requirement to stay at the office late and miss a favourite show could access their home VCR from the office and pro-gram it.

15 Telephone calls will eventually be digital and most likely use the Internet as the digital network. Like the VCR, this does not mean all phones would need a qwerty keyboard and colour display. They will both represent other classes of Internet connected devices- not requiring the
20 exact same configuration as PC's.

The present invention facilitates the production of a small, economical device which is particularly arranged to deal with communications, to build, compare and deconstruct message information. Such a device is novel maybe termed a
25 Specialised Network Access Computer (SNAC). The applicants believe that a SNAC could emerge as a class of device allowing data entry and control through the Internet where a smaller, more economical device than a conventional PC is appropriate. In a preferred embodiment, the device is
30 implemented utilising a virtual machine having a message processor and a protocol processor as discussed above. In the preferred embodiment, the software of the device can be considered to include three layers of virtual machine software (the HW drive layer, the Hardware Abstraction
35 Layer, and the Virtual Machine Processor layer) and a software application. All layers other than the Virtual

Machine Processor Layers are well established by prior art. A payment terminal can be used substantially without alteration as the hardware component of the device. A hardware abstraction layer (HAL) is a set of routines providing a common application program interface (API) to exercise the operating system, BIOS or hardware drivers.

HAL consists of routines to either (a) implement the functionality not provided by the underlying operating system, BIOS or hardware drivers, but needed for the common API, and (b) translation of parameters and adjustments of functionality required to adapted underlining OS, BIOS routines for the routines specified by the common API.

Such a SNAC can be applied in many different types of communication application over a network.

The present invention also facilitates the production of devices which incorporate a snac as a functional element of the device. Such devices could include both devices collecting information for transmission over a network such a pay telephones, particularly those equipped with smart card facility, or devices receiving information from a network such as the futuristic VCR or even washing machine.

Preferably, message instructions and protocol instructions may be developed on a convenient device such as a PC or general purpose computer, utilising a development tool in accordance with another aspect of the invention.

From a further aspect, the present invention provides a development tool for developing message instructions for providing directions for operation of a message processor means to be implemented in a virtual machine as discussed above, the development tool comprising a processing apparatus arranged to receive data input by a user to build message instructions for the message processor means.

The arrangement is preferably driven by a graphical user interface based program which provides various screens and fields for the user to input data relating to message

instructions.

The message instructions are preferably subsequently converted to code and downloaded into the device which is to employ them with the virtual machine. From a further aspect the present invention provides a development tool for developing protocol instructions for directing operation of a protocol processor means to be implemented with the virtual machine as discussed above, the development tool comprising processing means arranged to receive data input by a user to build protocol instructions.

The arrangement is preferably a program which is arranged to build protocol instructions from the data input by the user. The program is preferably graphical user interface based and provides screens and fields to facilitate data input for the protocol instructions.

Protocol instructions and message instructions can therefore be built on a PC and downloaded to device where the virtual machine is to be implemented.

A tool has also preferably been provided for developing function processor instructions, along the lines of the tool for the protocol processor instructions and message protocol instructions.

Limited hardware provided by such a device as a payment terminal or other SNAC device does not lend itself to development and testing of applications programs. Although the finalised application must run on the hardware, to develop and test an application it is more convenient to be able to utilise a more user-friendly device, such as a PC or general purpose computer.

From a further aspect, the present invention provides a communications device including a virtual machine means including a protocol processor means arranged to organise communications to and from the device and protocol processor instruction means arranged to provide directions for operation of the protocol processor means.

From a further aspect, the present invention provides means for emulating a virtual machine on a PC or other general purpose computer, the virtual machine comprising a message processing means and function processor as
5 discussed above. The virtual machine is arranged to operate on the PC or other general purpose computer so that instructions developed for the machine can be tested.

Similar emulation is preferably provided for the protocol processor means.

10 Emulation can therefore be used to test payment terminal or other SNAC application programs.

The present invention yet further provides a method of programming a device for processing communications, comprising the steps of loading a processing means of the device with a virtual machine which includes a virtual
15 function processor and function processor instructions for controlling operation of the device, and a virtual message processor which is arranged to be called by the function processor and which is arranged to carry out the task of assembling, disassembling and comparing messages, whereby
20 when a message is required to be handled by the communications device the message processor is called to carry out the message handling task.

The method of programming preferably also includes the
25 step of loading the processor means of the device with a protocol processor means arranged to organise communications to and from the device, and protocol processor instructions arranged to provide directions for operation of the protocol processor means.

30 The present invention yet further provides a computer memory storing instructions for controlling a computing device to implement a virtual machine means which includes a virtual function processor and function processor instructions for controlling operation of the device, and a
35 virtual message processor which is arranged to be called by the function processor and which is arranged to carry out

the task of assembling, disassembling and comparing messages, whereby when a message is required to be handled by the communications device the message processor is called to carry out the message handling task.

5 From yet a further aspect the present invention provides a computer readable memory storing code for implementing message processor instruction means arranged to provide directions for operation of a message processor in a virtual machine means, the message processor being
10 arranged to process messages for communication to and/or from a device.

 From yet a further aspect the present invention provides a computer readable memory storing code for implementing the virtual machine including a protocol
15 processor means arranged to organise communications to and from a device.

 From yet a further aspect the present invention provides a computer readable memory storing code for implementing protocol processor instructions arranged to
20 provide directions for operation of a protocol processor means arranged to organise communications to and from a device.

 From yet a further aspect the present invention provides a specialised network access computer, including a
25 micro processor and a virtual machine means, the virtual machine means including instructions for running on a virtual micro processor and an interface enabling the micro processor to operate the virtual processor.

 Preferably the specialised network access computer is
30 a payment terminal or other type of "card computer" (being a computer which is arranged to process information from cards and/or communicate information to cards - cards being smart cards, magnetic cards or similar).

 The interface between the actual processor and the
35 virtual processor preferably includes a hardware abstraction layer (AJL) or the like which provides a common

API.

Features and advantages of the present invention will become apparent from the following description of an embodiment thereof, by way of example only, with reference to the accompanying drawings, in which:

Figure 1 is a schematic block diagram of a payment terminal in accordance with an embodiment of the present invention;

Figure 2 is a schematic diagram of a control program architecture for the embodiment of figure 1;

Figure 3 is a schematic flow diagram illustrating device operation which requires the operation of the message engine;

Figure 4 is a schematic flow diagram illustrating an example of operation of the protocol engine;

Figure 5 is a representation of a display (screen dump) available on a development tool for developing a program for a device in accordance with an embodiment of the present invention, illustrating development of a message instruction for an example message;

Figure 6 is a screen dump of a further development tool display illustrating further detail of development of a message instruction;

Figure 7 is a further screen dump of a development tool display illustrating further detail of development of a message instruction;

Figure 8 is a screen dump of a further development tool display illustrating development of a further message instruction.

Figure 9 is a screen dump of a further development tool display illustrating development of a protocol instruction;

Figure 10 is a screen dump of a further development tool display illustrates further detail of development of a protocol instruction;

Figure 11 is a schematic diagram showing a structural embodiment of the message instructions and description for the message processing means, and

Figure 12A is a schematic diagram showing the
5 structure of protocol instructions for an embodiment of the protocol processor means.

Figure 12B is a representation of a display of a development tool for developing protocol instructions.

An embodiment of the invention will now be described
10 particularly with reference to a payment terminal device. The invention is not limited to payment terminal devices and the following description is given as an illustrative example only. The invention can be employed in all devices concerned with communications over a network, such as a
15 Specialised Network Access Device.

A payment terminal device in accordance with an embodiment of the present invention is illustrated in figure 1. The device hardware comprises a processing means which, in this embodiment includes a central pro-cessing
20 unit 1 and a memory 2 for storing instructions and data. The device further comprises a keyboard 3 for input; a card reader for inputting information from a card 5; a display 6; a printer 7, and a communications interface 8 for communication with an account acquirer.

25 Prior art devices generally have similar arrangements to that illustrated in Figure 1. The number and type of peripherals to the CPU may vary, but the essential operation required by the prior art and the present invention are similar.

30 Such devices operate to facilitate remote payment transactions, and a general overview of operation is as follows:

(1) Information is taken from an account holder's (customer) card 5 via a card reader 4. Transaction
35 information is input via the keyboard 3. The transaction information may include a money amount. The display 6 may

prompt the user (merchant employee, customer) to input information (e.g., it may ask a merchant employee to input an amount) and may also display information as it is input. The keyboard 3 may also be used by the customer to input a
5 code for the account, such as a PIN number.

(2) The CPU communicates the information via communications interface 8 with an account acquirer computer. The account acquirer computer may carry out a transaction (e.g., deduct money from the customers account
10 and pay the merchants account) or may provide an "authorisation" that a transaction can be carried out. Information that an account transaction has taken place or that the account acquirer authorises a transaction to take place is transmitted to the communications interface 8 from
15 the account acquirer computer. A display 6 may be provided to indicate that the transaction has occurred or may proceed.

(3) When the transaction is complete, a print out of transaction information may be provided from printer 7.

20 Prior art payment terminal devices are generally programmed in a conventional manner. That is, programming comprises a sequential set of operating instructions which are executed in sequence to carry out a remote payment transaction. This "sequential program" may be directly
25 compiled onto the processor of the device so that the device is under direct program control or, as is more usual, an applications program in a conventional programming language may control operations through a BIOS/OS. Whatever conventional programming form is used,
30 however, the device suffers from the problems which are discussed in the preamble of this specification. The programs are not portable between devices having different hardware or operating system architectures and it is necessary to write a program specifically for each type of
35 device. Further, any amendments to the operation of the device must be programmed by a programmer having knowledge

of that particular device and program arrangement.

Figure 2 is a schematic block diagram illustrating architecture of a device in accordance with an embodiment of the present invention.

5 The architecture comprises the hardware 100 the device, as illustrated and described in relation to figure 1. It also comprises the hardware drivers, known in the prior art, and including an existing BIOS/OS or hardware drivers, reference numeral 101 and also includes the
10 Hardware Abstraction Layer Interface (HAL) 102. The HAL 102 and hardware drivers 101 form a layer of a virtual machine which also includes virtual machine processors 103.

 The virtual machine 101, 102, 103 is arranged to emulate a hypothetical payment terminal. Application 104
15 controls the virtual machine 101, 102, 103 which in turn controls operation of the hardware 100. The virtual machine 101, 102, 103 can be adapted for many different hardware 100 arrangements (i.e. many different brands of payment terminal). Different arrangements of hardware 100
20 can therefore be controlled by the same application software 104.

 The provision of Hardware Abstraction Layers and hardware drivers for virtual machines is known in the prior art and fully described in various publications. Each
25 peripheral of the virtual machine is defined to be able to act in some manner on a standard set of commands. The HAL implements the best interpretation of each command on the actual peripheral present. For example a printer is defined to implement a "feed paper ready for tear off"
30 instruction. On differing roll paper printers this requires feeding a different number of lines, on tractor feed printers this requires feed to the next perforation.

 The virtual machine processors include a message processor 105 and a protocol processor 106, implemented in
35 software code. The message processor is arranged to process messages communicated to or to be communicated from

the payment terminal via the communications interface 8. The protocol processor is arranged to organise communications to and from the device, and to control and select the sequence of message processor operations in relation to messages received and transmitted. The message processor 105 and protocol processor 106 are implemented in native code of the payment terminal and therefore operate at relatively high speed. Because much of the "work" of the payment terminal is in building, comparing and deconstructing messages and processing communications, the operation of the device is relatively quick even though employing a virtual machine, 101, 102, 103.

The virtual machine processors 103 also comprise a function processor 107 the operation of which is to control and select general operations of the device not specially controlled by the message and protocol processors 105, 106. The function processor is also preferably implemented in the native code of the micro-processor of the hardware 100.

The application 104 includes protocol instructions 108, message instructions, 109, function support 110 and function instructions 111. The protocol instructions 108 govern operation of the protocol processor 106. The message instructions 109 provide directions for operation of the message processor 105. Function support 110 and function instructions 111 govern operation of the function processor 107. The application 104 and virtual machine 101, 102, 103 operate on data 112 input to the payment terminal to process it in accordance with the application 104.

In this example, the application include a set of "primitives" which are a series of symbolic commands which are executed by the device to control carrying out of a remote payment transaction. The appendix A to this specification lists primitives utilised by a preferred embodiment of the invention and gives descriptions of their

respective functions. It will be appreciated, however, that a skilled person would be able to design their own primitives for carrying out remote payment transactions and the invention should therefore not be considered limited to use of the primitives listed in the appendix. It is in fact anticipated that users of the system may desired to created their own primitives and product documentation attached includes instruction for this procedure should it be desired.

10 Appendix A is in the form of a "HELP" file to be used with a product. The important information for the purpose of this description is the brief description of each "PRIMITIVE" and their function.

15 The primitives operate utilising the data 112. The data 112 may be data being input to the device, such as the customers account number, information which is fixed (strings) in the device e.g., a particular account acquirers identity.

20 The function processor 107 includes an event scheduler and index as known in the prior art. In response to an event (e.g., swipe card) the event scheduler operates via the index to look up a sequence of primitives 11 to be executed in response to that particular event.

25 In the preferred embodiment, the virtual machine processors 103 are constructed using C and the application is constructed using C++ or Java.

30 The device of this embodiment is event driven. When converting a device incorporation the SNAC hardware requirements to a SNAC by the provision of an appropriate HAL and virtual processors, and event driven structure can be added to a non-event driven underlying architecture through the HAL. This can be achieved through a software loop detecting events and generating an event call for any detected event.

35 The application 104 responds to the occurrence of an event to dictate subsequent operation of the de-vice. For

example, when a card is swiped through card reader 4, the appropriate sequence of instructions from the application 104 will be implemented. The event driven structure allows the hardware drives 101 to have control during idle periods. When an input event occurs the application is called to process the input and then returns control to the hardware drives 101.

The application may be loaded on a remote payment terminal device with a pre-existing operating system. Where the operating system is event driven HW drivers 102 can operate as an interface layer without any problems. Where the pre-existing operating system (HW drivers) is not event driven, amendments must be made via the HAL to convert to an event driven structure.

Appendix B includes a description of a operation of the HAL 18 in accordance with an embodiment of the present invention, on a functional level. A skilled person would be able to develop an appropriate HAL structure for an existing device or a new device. The appendix B is in the form of a "HELP"-file for a product. It merely describes an example of implementation of a HAL and adaptation of an existing devices existing BIOS.

Figure 3 illustrates an example of an operation of the device, for one typical step in a remote payment transaction. The other steps in the remote payment transaction are carried out in a similar way. That is, they may require the operation of the message processor 105. They are event driven, such that the application 104 is called up to deal with any particular event after the event occurs, etc.

The operation schematically outlined in figure 3 is that of reading information from a customers card and storing information in fields for subsequent processing by the application 105. In overall operation of the device, the information from the card will be required to identify a user and enable access to the user account to cause a

transaction or authorise a transaction.

Figure 3A illustrates example information included on a magnetic stripe on a magnetic stripe card 5. The information includes track 1 information, track 2 information, track 3 information, the customer name, the PAN, the expiry date and End-Of-Form label. This information must be taken off the card and stored in appropriately labelled fields so that it can be accessed to enable processing of the transaction.

10 At step (1), on a card swipe of card 5 through reader 4, the card swipe event is detected by the HW drivers 101.

The HW drivers 101 causes a call back to an event table in HAL 102 for the peripheral card reader 4 which contains a series of names for routines to be performed on the occurrence of a particular event on the card reader 4. There are also event tables for the other device peripherals.

Figure 3B is a schematic illustration of the event table for the card reader 4. Event "2" is for card swipe. In this example, there are three alternatives available for a card swipe event, labelled "1", "2" and "3". These labels may be dynamically updated in the event table, depending upon the particular stage of operation of the device.

25 Label "1" is for the routine "handle idle card". This is a routine which is instigated where no payment transaction routine has yet been instigated, i.e., this is "kicking off" operation.

Label "2" is the label for the "handle card" routine. This is where the payment terminal device is waiting for a card read event, e.g., where one has a device of the type which requires a money amount to be input before the card is read.

Label "3" is where the device may be at a stage in the operation where it does not require a card reader, i.e., the card is swiped in error. In this case, nothing happens

and no routine is initiated.

Note that the above descriptions of the routines are not "primitives" but are merely general descriptions.

It will be appreciated that the event table may
5 contain labels for any number of events to carry out
operation of the device peripheral the card reader 4.
Similarly the other event tables for the other peripherals
will be configured with labels for various routines they
are required to carry out, as will be appreciated by the
10 skilled person. It is not necessary to go into detail
detailing all the routines, as they will vary from device
to device and will be a matter of choice of the skilled
programmer, and the operator of the payment terminal
device.

15 This event table driven structure is ideal. In a
conventional terminal, where the terminal is executing
sequential program instructions, for "handle card" routine
the device will merely sit in a loop waiting for a card to
swipe. With this architecture, however, the device does
20 not have to sit in a loop waiting for a card swipe. It can
leave the application program and return to the HW drivers
101 and in the mean-time the CPU 1 can be carrying out
other jobs.

With the event label, the sequence of the application
25 instructions for the particular routine is then looked up
via an index from the application 104. The function
processor 107 is then called up, step (3) to commence
implementation of the instructions for card swipe. The
function processor 103 then implements the instruction
30 sequentially. The function processor 103 is a conventional
interpreter, as will be understood by those skilled in the
art, arranged to implement the high level primitives of the
application 104 via HW drivers 101.

The first primitive requiring execution for the
35 "handle card" routine in this example is the SAVE
primitive, step (4). The first operation of the SAVE

primitive is to call up the message processor 105. The message processor 105 is a series of several sub-routines implemented in the native code of the CPU 1, the specific operation of which is to construct, de-construct and
5 compare messages in accordance with message instructions 109 from the application 104. The SAVE primitive will have associated with it a label indicating the particular message instruction 109 associated with this particular event. The function processor 107 fetches the message
10 instruction 109 for this event and the message processor 105 then operates to load the data from the card into labelled fields (steps 5, 6 and 7) according to the message instructions.

Once the message processor 105 has loaded the
15 information from the card into the appropriate fields, in accordance with the message instructions 109, the SAVE function is completed and the device proceeds to carry out the next function in the sequence for "card swipe" fetched by the function processor 107. Alternatively, the sequence
20 of functions for "card swipe" may be completed and the device may wait for the next event before proceeding further.

There are a number of ways that the payment transaction could continue once the SAVE function has been
25 carried out. For example, steps could be taken to create a display asking the customer to input a PIN. Again, such steps would be carried out by the function processor 105 implementing the instructions, which would include a function to call up the message processor 105 to build a
30 "form" to display the request on the screen. Alternatively, the device could be controlled to take steps with regard to the information loaded into the fields by the card in accordance with the SAVE function. For example, it could compare a PAN number taken from the card
35 with an equivalent PAN number stored in memory of the device to establish the identity of the account acquirer.

A skilled person will realise that a number of possibilities are available for continuing with the transaction, and would be able to formulate appropriate programming from this description and the following appendices.

As discussed, the message virtual processor means is directed by message instructions 109.

Figure 11 is a schematic diagram illustrating the structure of the message instruction means 109. The message instruction means is in fact in the form of a set of "descriptions" of the messages. Each message usually comprises a plurality of fields 120, and the message instruction means for each message contains a corresponding plurality of message instructions.. One field may be the CUSTOMER NAME, for example. In the message instruction means, each field is associated with a number of message descriptors 121 which designate characteristic to be applied to the information in that field or to be expected of the information in that field. Operations which may be carried out on the data included in that field may also be included in the descriptors 121. As illustrated in the drawing, the descriptors may include:

1. Data Location Identification. This will indicate either where the data is to be found and/or where data is to be put. In the current embodiment the data location information is contained in a two byte field descriptor (thus having 65535 different possible values) with value ranges allocated to

- 1) 2000 strings
- 2) literal numeric values from 0 to 32,000 in abbreviated form
- 3) data field IDs where each ID is represented as an entry in a table, and each table may contain up to 256 fields.

2. Data Representation (i.e. Ascci, Binary, etc.). This indicates what representation form the data is in

and/or what it is to be converted to.

3. Format. This provides a description of the format that the data is in and/or is to be placed in.

4. Test Function. The index of a function processor set of instructions to determine if the current field is to be included or excluded at this time

5. Line & Column. Relative position for use in constructing messages for display or printing. These values are used to determine the quantity of space characters, and or new line characters that are required in the buffer.

6. Substitution list. A list of text representations to substitute for numeric values e.g., display the value "1" as "Monday" and "2" as "Wednesday".

7. Additional description options as required by the application or prove useful in future embodiments.

Each message instruction will therefore include a description of a field of message data, providing instruction for the virtual message processor means which enable it to carry out a number of tasks:

1. To compare a message with a message description to see if it is the correct required message.

2. To take a message of the correct description from a location and place it in an-other location.

3. To take a message and deconstruct it into various components and place the various components into other locations.

4. To take data and build a message in accordance with the message description and place the built message in a location.

5. Compare one message with another message.

Other functions may also be carried out by the message processor as required by the application. The message processor can manipulate data in any desired way in accordance with descriptions provided by the message instructions. Messages comprising data can therefore be

billed, placed in locations, taken from locations, de-constructed with elements being placed in locations, etc. for subsequent operation on the data by the application. Any device which deals with significant amounts of messages
5 in such form can therefore benefit from this arrangement.

Each message description is labelled so that it can be identified by the application, e.g. each message description may be numerically labelled.

A development tool for developing the application 104,
10 in particular the message and protocol instructions 108, 109 comprises a graphical user interface based program which may be run on a PC or other general purpose computer. The program provides a graphical user interface based
15 from data input by a programmer. Message instructions can subsequently be translated into code readable by the virtual machine 102, 101, 103 and downloaded into the application device. Figures 5, 6, and 7 are
"screen dumps" which illustrate displays generated by the
20 development tool for an example message instruction. In this case the message relates to data from a magnetic stripe of a customers card. The message instructions direct the message processor 105 to take the fields of the message and place them in known locations in accordance
25 with the instructions. Such a message instruction may be called up in response to the SAVE primitive, in the event of a card read. Data from the magnetic stripe of the card would be stored away in the appropriate locations in accordance with the instructions, for subsequent
30 processing.

Each message is provided with a message name 30, in this case "TrackData". This message name identifier can be used to call up this particular set of message instructions in the development tool. An alternative
35 numeric identifier is generated for use by the virtual processor. This numeric identifier may also be displayed

by the development tool. Each message is made up of a number of message "fields" 120. In this particular example, there are seven fields, being "Track1", "Track2", "Track3", "CustName", "PAN", "ExpD" and "End-Of-Form".

5 Each of the seven field is converted to a message instruction for use by the virtual message processor. This is the information which is typically found on any magnetic stripe card. The message instructions in accordance with this embodiment direct the message processor to process
10 these elements. Each field is associated with descriptors which provide further instructions for the handling of that element. Figure 6 illustrates a display 33 which enables a programmer to provide message descriptors to CustName element.

15 Each field 120 has a "format" descriptor 34.

There is an instruction as to the Data Representation ("Type") reference numeral 35. In the illustrated embodiment there are four types, Asc11, Hex, Binary and BCD. There is also a logical operation instruction (option
20 test), reference numeral 36. This logic instruction can be used to determine whether or not the message processor will process this element at all, for example, i.e., it will only include the CustName element in the message when the logic function equals "True". Other instructions designate
25 the data source, reference numeral 37, in this case a field, and the field label, reference numeral 39. The format 34 is labelled with a name, in this case, "Tracks". There are further instructions which dictate the format Tracks to be applied to CustName. Figure 7 shows a display
30 which illustrates the instructions for the format "Tracks".

The message processor is responsive to all the message instructions to load the data from the magnetic stripe card into the appropriate fields with the appropriate formats in accordance with all the rules designated in the
35 instructions.

This embodiment of the present invention includes another class of message instruction means, known as a "Form". Instead of a Data Representation as a message descriptor, a Form includes description of a Location of the data field in the Form. Figure 8 is a display provided by a development tool enabling the programmer to prepare message instructions for a Form message. On the left hand side of the display a panel 70 illustrates Form layout. The fields in the Form include MerName, Address Line 1, etc. The location of these fields can be moved within the panel 70. The location in the panel is provided as a descriptor and for the message instruction. The Form type of message instruction controls displays, reports, print-outs, and the like. The type of Form is given by the instruction designated by reference numeral 71, in the example illustrated in figure 8 being a print-out. The message processor takes the fields from known memory locations or other locations and enters them in the locations enabling the Form described by the Form instruction to be produced.

As discussed previously, another major function of a SNAC device is communications. For example, it is necessary for the majority of remote payment transactions for communications to be able to occur between an account acquirer location, in order to enable access to an account, and the remote payment device. Communication with a data carrier, such as a smart card device may also be required.

The protocol processor 106 is arranged to organise communications, in accordance with directions from the protocol processor instructions 108. Referring to Figure 4, in a typical remote payment transaction, after a card has been swiped, a PIN number has been input and a charge amount has been input, information then needs to be communicated to an external computer, at the account acquirers, in order to enable further processing of the transaction. After an event such as a communications

message arriving, therefore, HAL 102 detects the event
(step 1) and activates the protocol processor (step 2),
figure 4. The protocol instruction 108 for the event is
rolled up (step 3). The protocol processor 108 implements
5 the protocol instructions for that event, (step (4)).

The protocol instructions are divided into "sections"
130, "lines" 131 and "protocol commands" 132, as
illustrated in Figure 12A. Figure 12B illustrates how an
instruction is displayed on a development tool for protocol
10 instructions. Protocol instructions describe message flow
both from and to the device. The top line specifies
outgoing messages and the other lines display possible
incoming results. A protocol consists of lines and
sections. At the start of each section is a line 1
15 (optional for the first section) which describes the
outgoing message. There are a number of protocol commands,
and these include:

1. Protocol - Run a sub protocol
2. Message - Send a message or handle an
20 incoming message using the virtual message processor means
3. Retry- re execute the steps of protocol from
and indicated point
4. End - End of the protocol
5. Exit - Stop the protocol from an
25 intermediate point
6. Timeout() - Specify the a delay after which
the protocol should automatically jump to the point at
which the timeout instruction is placed.
7. Control - Specifies a control character to
30 be send or received.
8. Function - Execute a virtual function
processor function

Protocol instructions are organised in lines and
sections. In each section Line 1 indicates the information
35 to be send by the SNAC device and subsequent lines indicate
actions to be taken in response to the alternate possible

events which may occur in reply. The first instruction on each of these subsequent lines is used to identify the response. Control(), Message(), Function and timeout() may all be used to identify responses as follows.

5 1) When the time specified by a timeout instruction elapses then the line commencing with the timeout will be selected.

 2) When data is received it will be sequentially compared to a lines commencing with Control() Message() or
10 Function to see if the data matches the control character, matches the message of causes the test contained in the function to evaluate to true.

Figures 9 and 10 illustrate displays of a development tool for protocol instructions for the protocol "General"
15 which is the Protocol Name (reference numeral 42). Instructions are presented as a screen dump in the form of a table 43, which can be accessed by a programmer if he wishes to alter the protocol.

Protocols are arranged to control message flow both
20 from and to the target device (e.g., account acquirer computer). The top line of the display panel 44 specifies outgoing messages and the other lines display possible incoming results.

A particular protocol is able to call up other
25 protocols "nested" within it and is also able to call up the message engine to deal with messages.

Referring to figure 10, the top line of panel 44 specifies the outgoing message. The first operation of protocol "General" is to call up and carry out a further
30 protocol, "Reversal". Figure 11 illustrates instructions for the protocol Reversal, reference numeral 45. Reversal operates to call up the message engine to construct message number 0400 and this message is then sent to the target device.

35 The either

1) Message number 0410 should then be received back from the target device and the message processor will be called up to deal with that data, which involves the message processor comparing the incoming message against the description specified by the message instruction means and storing the data if a match occurs. Or

2) A timeout of 100 tenths of a second elapses.

Then the protocol is ended and re-turned to the protocol General, which causes a further message, 0100 to be formulated and sent out.

Then either

- 1) A message matching 0110 will be received or
- 2) A message matching 820 will be received or
- 3) neither 1 or 2 will occur for the timeout() period, in this case specified as 000 tenths of a second.

If the message 0110 should then be received from the target device and compared by the message engine, then another protocol "adjustments" will then be carried out. The protocol would then end.

If the message 820 should be received from the target device, which can be dealt with by the message engine and compared with the instructions from the message instruction means. The "Retry" instruction will then be executed causing the virtual protocol processor to move execution back to the sending of the (0100) message. The retry count of zero indicates this loop would continue whilst 820 messages are received.

If the Timeout occurs, then the retry(5) would be applied causing the protocol processor to move execution back to the Send 0100 message. This loop would occur up to five times as indicated by the retry(5). After the fifth time execution would move to the next section causing the protocol to End.

More details of operation and build up of messages and protocols are given in the appendix A.

The device in accordance with the present invention,
for example a payment terminal, may be implemented in GAVA
by defining a class library payment terminals. This class
library would contain calls to all the functions of how HAL
and preferably the message and protocol engines.
Similarly, a specialised network access computer or card
computer could be implemented in GAVA.

Please note that the arrangement of the present
invention can be used to deal with any payment transaction
device, including one which deals with smart cards.

The present invention can also be used to implement a
specialised network access device, which may use similar
hardware to that provided for a payment terminal.

In the attached Appendix A, the term "CardScript" is
the name the applicants have given to programming required
to implement this embodiment of the invention.

It will be appreciated by persons skilled in the art
that numerous variations and/or modifications may be made
to the invention as shown in the specific embodiments
without departing from the spirit or scope of the invention
as broadly described. The present embodiments are,
therefore, to be considered in all respects as illustrative
and not restrictive.

APPENDIX A

Contents.

Introduction

Introduction

Help for CardScript Scribe

The Scribe program assist in the design of stored
information & programs for EFTPOS terminals, PINpads,
Electronic Cash Registers and other small computer systems.

Writing A Program

For help on writing a CardScript, program, rather than
5 operation of the Scribe tool, see

Writing A CardScript Program

A CardScript program is more similar to a Windows RAD tool
10 program than a conventional C Language or Assembler program.

The "target" device has several keys, one or more card
readers, and usually one or more communications ports.
Defining a program consists of attaching actions to these
15 events, or the special events of terminal power on and
terminal idle.

CardScript programs - as all other program - manipulate
data. Data is defined in a Data Dictionary. Unlike normal
20 programs, it is normal to write many CardScript programs
using the same Data Dictionary. Standard Data Dictionaries
are available from CardSoft for EFTPOS and several other
application types. It is recommended to write initial
applications based on one of these standard dictionaries.
25 Once the program is experienced, the Data Dictionary for an
Application may be modified. see

Configure Data Dictionary

30 Data Dictionary Usage

The Data Dictionary represents the list of all "variables"
or information values used in the target device. These
"variables" are in formation which may change over time, or
35 be different from device to device.

Information which is fixed for all devices usually is defined by strings. All information to be included in displays, receipts, messages etc, comes from either the Data Dictionary or from STRINGS.

5

Data Dictionary fields may have an initial value set from the Initial Data Tables

Structure

10

Tables

The Data Dictionary is divided into tables. Each record displayed in Configure Data

15

Dictionary describes one table. Fields are placed by selecting add and clicking on the Panel.

Field Attributes

20

Double Clicking on any field reveals and allows viewing and/or editing of Data Dictionary Field Attributes.

Field Order

25

Layouts are stored indexing fields by table#lfield#. This means existing scripts will behave strangely if the Data Dictionary is changes the number Of referenced fields.

30

For example if "Merchant Name" is table 3/field 2 and "Address" is table 3lfield 3. Then deleting field table3/field1 will make any prior references "Merchant Name" now reference "Address". This can be remedied by inserting a dummy table3/ field 1 as a placeholder.

35

Generally this problem does not arise since new dictionaries are not to be used for old applications, and existing dictionaries are usually only extend. In the rare

event that a dictionary used by existing applications is to have fields deleted, it recommended to rename them to "dummy" or "unused".

5 Be careful since any existing data in the files will be rearranged when retrieved, it will simply be move from the record into the fields in the order listed at the time. New fields added in graphic display mode are always added at the end

10 Reserved Settings

see Reserved Data Dictionary Settings

see also

15 Data Dictionary Field Attributes

The field attributes which may be set are as follows

20 Type

Type refers to the format in which data is held. "X-Ref" is a special value used to indicate that another table will be referenced at run time and thus must be included in the
25 build.

Binary Data Fields

30 Binary. either 1 or 2 bytes in length for Integer values in calculations, longer fields hold bit fields or keys. 250 bytes is the maximum permissible number of bytes

Maximum Integer values

35 Depending on the number of bytes used to represent the Binary number, the following values are possible

- 1 Byte - 0.. 255
2 Byte - 0.. 65,535
3 Byte - 0.. 16,777,215
5 4 Byte - 0.. 4,294,967,295
- Text - up to 250 bytes
BCD up to 250 bytes
Date / Time (2 Bytes for Dates, 2 or 3 bytes for Time)
see Date & Time Fields
- 10 Amount - 10 bytes, internal format is target device
dependent
Packed Amount - not currently used
X-Ref - Advanced use only
- Flags
- 15 0 = Field is fixed and never reset
1 = Reserved for future use
- 20 2 = Reserved - used with deleted fields
3 = Field is reset when terminal is loaded
4 = Field is reset at power on
- 25 5 = Field is reset by idle function

Bytes

- 30 The length of stored data in bytes

Length

- Caution: When you create new data dictionary fields, make
35 sure their length is not zero if you want to use them, or
they will be invisible.

The number of characters allocated to display the field as text

5 Name

The name of the field for display on receipts etc.

Table

10

The "refer" Initial Data File from which the field initial value is extracted. Blank if the field is extracted from the default file.

15 Table ldx

When "Table" is non-blank, "Table ldx" specifies the "refer" of the Initial Data Field in the default file used to indicate the record number in "Table" from which data is to be extracted. In other words the join field between the default table and the joined table.

20

Field

25 The "refer" of the Initial Data Field from which the initial value of the field is to be obtained.

Creating a New Application

30 As suggested above to create any application, it is recommended to copy a "template" application. Simply copy the entire template directory.

To then work with the new directory, select

35 File/Installation and edit the data directory. (Don't forget the trailing I) Is the recommended to exit Scribe

and restart. Scribe and restart.

Console/Display

- 5 The display console used with CardScript is quite sophisticated. see

The Console

- 10 CardScript can be used in a variety of devices, some of which may not support all the features described here.

Features

- 15 The CardScript Console has a number of sophisticated features

Hot Keys

- 20 Keys used to launch only one action, where the action is part of the application, are known as hot keys. Typically the action may be activated only when the terminal is idle. For further information see the "KeyBd" primitive.
- 25 In an EFTPOS application, on a terminal with Keyboard Buttons available for allocation, Hot Keys will normally be allocated to such functions as "Sale", "Adjustment", "End of Day" etc.
- 30 Hot keys normally would have their label printed on the keyboard, or on the physical button.

Multiple Field Input

- 35 On any Layout displayed on the console, several field may be selected for input. The OK key steps from one input to

the next. Any soft key terminates all input.

Scrolling

- 5 The display may be scrolled, permitting a larger virtual display than the physical display. Scrolling is performed automatically by the driver in the target device. All that is needed to enable scrolling is to tell the scrolling driver what keys on the keyboard perform scrolling. The
10 keys used to scroll are set by the

Console Primitive

Console(Command,Parameter)

15

The command determines which of the following console options is set.

Command 1- Set Scroll Keys

20

The Parameter is a string of four hex values, in order - key-left, key-right, key-up, key-down

- The keyvalues specified are assigned to the scrolling
25 engine within the target device. Note scrolling my not function on all CardScript "targets" and the size of the scrollable area may vary.

Command 2- Set Keymap

30

The parameter is a Key board map. This command is now the preferred method for setting the keyboard map. The old Keymap primitive will be obsolete in a future version. See

35 KeyMap

Structure of Keymaps

The "field" or String used as a keymap must be a list of
5 4digit hex blocks, the first two digits of each block
representing the hex code of the key to be mapped and the
next two digits representing the hex value to map be
returned. Usually used from the startup function, using a
field from the terminal groups record.

10

Note that the following key codes have special meaning.

08

Represents a backspace or 'CLR' code.

15

0A

Represents an 'OK' or 'Enter'

1B

20 Represents a cancel.

30 through 39

Represent the digits "0123456789"

25 Command 3 -Keybd

The parameter is evaluated to zero, or non zero.

30 Upon entering idle state, the action of the keyboard is
determined by the last KeyBd command. The keyboard (except
cancel) will be ignored if off was specified.

This command replaces the old keybd primitive, which will
be obsolete in a future version.

35

Command 4- Invalid Entry

The parameter is the text message to be displayed.

- 5 This command is designed to be called from an input validation function. Calling this command indicates the input is invalid, the text specialised in the parameter should be display to indicate the error.

10 Soft Keys

- Some keyboard buttons on a target device may be used as soft keys. As opposed to Hot Keys these are buttons which may be used to initiate different actions, depending on the display present when they are pressed.

- 15 Since the principle of Soft Keys is to use the same buttons for different actions, displays must in some way indicate to the user the operation of each currently active soft key.

20 Soft Key Button Sets

- Target devices may allocate certain buttons on the keyboard for use as soft keys. Button sets are numbered from zero. If a specified set is not available, then set zero is used. By convention:-

- 30 Set 0 = keys'O'thru'9' (the numeric keys)

Set 1 are dedicated soft keys, usually positioned directly adjacent to the display, in order that the display may be easily used to indicate their function.

- 35 Set 3 is the new standard for dedicated soft keys - hex values 81,82 .. AO.

Set 3 will normally be requested on forms where numeric/text input is also required.

5 Set 4 is the same as set 3, but allowing use of the numeric keys if no dedicated soft keys are available. Set 4 should not be specified on screens where numeric input is also available, since this may cause a conflict.

Soft Key Action Groups

10

These are the groups of actions that may be offered at any given time.

15 In Layouts/Forms, a soft key action set may be selected for any display. Individual functions may be assigned to an action group from the Function/General Purpose Functions, in the Function activation section.

20 Note Action group 0 is used to indicate a function is NOT part of any group

Correlation of Actions to Buttons

25 If a display allows soft key Button Set 0, (Keys '0', '1', '2', '3' etc) and action set "I" then when the '2' key is pressed, then soft key group '1' , An Group '2' (Key '2' minus the first key in the action set equals 2), if it exists, will be activated.

30 The Keyboard

To control the features available, and make best use of your terminal, you can recode the keys on your keyboard using the keymap primitive. This allows you to customise
35 the target device to allow portable and easy to operate applications. See

Keyboard Codes

The Keyboard codes are designed to accommodate a wide
5 variety of keyboard configurations. At any given time each
key (or button) will act as one of the following key types

- 1 Control/Data Entry Control
- 2 Data Entry
- 10 3 Soft/Hot key function activation

Control/Data Entry Control

Some keys are required for control. Control keys should
15 not be used for any other purpose than control, otherwise
the user interface will be incredibly confusing.

Minimum Requirements

- 20 All CardScript drivers should provide these key codes
without any mapping required.

- 08 Backspace or Cir
- 1B Cancel
- 25 0A OK or Enter
- 1A Fn - For general Function selection, and for double (or
triple) zero.

Additional Options

- 30 0D or complete form (combined with tab as an alternative to
0A)
- 09 Tab (move to next field- does not complete form)
- 0B Vertical Tab - Used as back tab or move to prior field.
- 35 1 1 (XON) DCI - Used as cursor left

12 DC2 - Used as cursor right
13 DC3 - Dedicated double zero
14 DC4 - Dedicated Function Key (combined with DC3 as an
alternative to 1 A).

5

Data Entry

Three levels of data entry may be available at any one
time. Text, Hex, Alpha. The bios can automatically
10 determine the available level and act accordingly.

Minimum Requirements

15 30.39 (Numerics)

Additional Options

A B C D E F (allowing Hex data entry)
Full 'querty' keyboard

20

Soft & Hot keys

Soft keys previously were recommended to be 'a"b-c' etc
Now the are recommended to be hex 81 82 83 etc up to a
25 maximum of AO allowing up to 32 dedicated Soft keys. The
change in recommended values is to allow for terminals with
full alphabetic keyboards.

Hot Keys (When/Additional to soft keys) should be allocated
30 from A1 ... D0

Program Portability

Portable Programs
35

CardScript allows the writing of totally portable programs,
it is also possible to write programs that are not very
portable. Any CardScript program will "execute" on any
CardScript enabled target, however the result could be of
5 no use on the target if special hardware characteristics
are required for practical operation of the program.
CardScript provides a mechanism for avoiding the traps and
keeping programs portable whilst still taking advantage of
special hardware when available.

10

Keyboard Traps

The key map primitive represents a trap in that this
function should never be used with a literal string, or
15 your program won't be portable.

Processing Cards

Magnetic Cards

20

Automatic Processing

Automatic Magnetic Card Processing, from

Upon Card read, data from the card is placed in the Receive
25 buffer. The format in the buffer is

Track 1 (I terminated)

Track 2 (I terminated)

Track 3 (I terminated)

30 Customer Name (I terminated)

PAN (I terminated)

Expiry Date (6 bytes ASCII)

If the read occurred at terminal idle, the Calculation
35 Result is set to zero and the system event Magnetic Card
Read is generated. After executing any function for the

Magnetic Card Read Event, if the Calculation Result is non zero this value is used to select a function for further processing of the specific card type.

5 Automatic Magnetic Card Processing

Upon card swipe, the data dictionary fields Transaction/Track2 thru Transaction/Customer Name are filled with the card details. These are data dictionary fields Table1/Field2 thru Table1/Field7. see Reserved Data Dictionary Settings for details.

Table 5 is then scanned to find a column matching the PAN of the swiped card. If a column in table 5 is found then the tables 3 & 4 have their columns set according to the entries for Issuer & Acquirer in Table 5.

Table 5 is set during the build to indicate the appropriate action should a card be swiped at idle. If the terminal was idle at the card swipe this function is now executed.

Typical Processing.

For standard processing, create a function as follows

```
store(O,CardMsg)
if ColFind(Pan,PanLow,PanHigh)
    ColSelect(Issuer, IssuTbl)
    ColSelect(Aquirer,AcqTbi)
Exit(O,CardFunc)
```

Smart Cards

Two primitives are available for controlling Smart Cards.

Card(Command,Field/Value)

1A command of 1 is used to read the smart card status into the field "FieldNalue". Using a value for "FieldNalue" does achieves nothing.

5

2A command of 2 is used to control the power to the card. If "FieldNalue" is 1, the card is powered on, if "FieldNalue" is 0 the card is powered off.

10 3Select. A command of 3 is used to select which smart card reader(or plug in is currently selected. By convention, 1 is the user card(or if only one reader is present, this reader), 2 is the separate merchant card slot or the plug in, where present. "FieldNalue" contains the card number
15 to be selected.

4A command of 4 is used to read the Smart Card Type Code

5A code of 5 performs a logical test on smart card type.
20 If the field/value supplied matches the Smart Card Type Code of the current code, the logical true flag is set. This command is designed to be used in an "if"test

6Code 6 reads the CardEntryMode into the specified
25 FieldNalue. See CardEntryMode 7Set Card entry mode to the value specified in FieldNalue

see Also

30 TPDU(Command, SendMsg,RxMsg,Status)

The TPDU primitive is used to send a command to the card. If the TxMsg is present this data is also sent to the card. If the RxMsg is present then a response is expected from
35 the card and is stored in the RxMsg buffer.

Command

This is the actual 5 bytes TPDU to be send to the card

5

SendMsg

This optional parameter specifies the message used to build the data send to the card.

10

RxMsg

This optional parameter specifies message used to store any data returned by the card in response to the TPDU.

15

Status

This mandatory message specifies the location of status variables to record the status of the TPDU operation.

20

The TPDU primitive with Synchronous (Memory Cards)
416 Cards

Drivers for 416 Cards support the following TPDU Commands

25

ReadBytes

WriteBytes

EraseBytes

Present Key

see Commands for Memory Cards

30

The present Key uses the length indicator to select either the CardSecret Code (2 bytes) or the application erase secret code (4 bytes)

35 Answer to Reset, & Card Type

The 416 has a card type code of 4 and an answer to reset of

3Bh 00 00 00 00 00

5

Commands For Memory Cards

ReadBytes

CL (any)

10 INS BO

ADDR XXXX (Byte Address an Card)

LN LL Number of bytes to read.

15

WfiteBytes

CL (any)

20 INS DO

ADDR XXXX (Byte Address an Card)

LN LL Number of bytes to write.

25

EraseBytes

CL (any)

30 INS DE

ADDR XXXX (Byte Address on Card)

LN LL Number of bytes to erase.

35 PresentKey

CL (any)

INS 20

5

ADDR XXXX (Byte Address on Card)

LN LL Lenght of Key.

10 Other Smart Card Commands

For selecting the smart card reader, and control of the reader.

15 For sending information to the card, and receiving information from the card.

Smart Card Type Codes

20 1 Async ISO type Card

2416

Asynchronous SCHLUMBERGER type EE2K

Asynchronous SCHLUMBERGER type EE4K.

Asynchronous SCHILUMBERGER type EE16K.

Asynchronous type GPM256

25 Asynchronous generic type 12C BUS

Asynchronous type GFM2K

9 synchronous type GFIV14K

For coding of smart card types.

30 Running A CardScript Program

To run the program on the PC simulator, see

PC Simulator

35

There is a implementation of the CardScript Virtual machine available on the PC that not only runs your program, it also emulates the keyboard layout and other controls of any target machine.

5

To run the simulator - select "Build/Run Simulation of Build"

Stored Information - Data Tables

10

For Information on setting & changing values in the Data Tables See -
Tables Menu

15 CardScript includes all tools for maintaining data tables to control the set up and distribution of Data Tables required for any application.

The System Data Table

20

The system data table has a fixed format identical in all systems. This table contains general information and current setting for use within the scribe program.

See -

25 System Table Settings

Settings in the system table are used to both miscellaneous settings in the script, and options for viewing the script.

30 Loader Settings

Terminal For Mask Load

Not used by scribe

35

Default Prompt (or String) Table

The string table displayed within Scribe. Multiple string tables may be used to support multi language applications.

5

Peripherals

Description the peripherals of a the target system here and displays and receipts will in scribe will show guides to assist in design. These setting have no affect on program execution in target devices and may be changed at any time.

10

Reserved Functions

15 Idle State

Set this pointer to indicate a function to be executed each time the "target" becomes idle.

20 Abort

Normally left at <none> since applications may vary default options during execution.

25 Initial

This function is executed at "target" power on.
Processor

30 Previously used to indicate the byte order used in the "target". It is now recommended to use "Low-High (INTEL)" for all systems.

Configurable Data Tables

35

The data tables used by CardScript can have their names, field names, layout and even the number of tables used altered according to the current system set up.

5 Configure Initial Data
 Initial Data Usage

The purpose of initial data tables is to provide a database of information for initial values for the data dictionary loaded into the target device.

Configuration

15 Structure

Each record in the configuration describes one table. Fields are placed on the Panel and dragged to the appropriate position.

20 Field Attributes

Double Clicking on any field reveals and allows viewing and/or editing of Initial Data Field Attributes.

25 Field Order

Clicking on "Graphic Display" toggles between the standard graphic view of the fields and a simple ordered list of the fields. In the ordered list mode fields may be dragged to rearrange the field order.

Be careful since any existing data in the files will be rearranged when retrieved, it will simply be move from the record into the fields in the order listed at the time. New fields added in graphic display mode are always added at the end.

Reserved Initial Data Base Settings

5 Certain fields must be present in the for the build
process.

File Usage

10 File 1 - "Terminals". The name may be changed however this
file is used to initialise individual target devices with
the optional "NetMgr" module. No other special usage at
present

15 File 2 "Groups" - no special considerations
File 3 "Issuers" no special considerations
File 4 "Aquirers" no special considerations
File 5 "Card Ranges" - must be used as card ranges, and
must have fields "lo" "hi" and "Ien"
File 6 "Products" no special considerations
20 File 7 "Region Settings" no special considerations
File 8 "Issuer Sets" no special considerations
File 9 "Card Sets" must contain the fields "cards" as an
index into card ranges
see also

25

Initial Data Field Attributes

Type

Flags

30

Current usage

0 = Place label to Left

I = Place lable above

35 Repeat

The number of times the field is to appear on the form

X-Pos

5

The current value of X-position of the field on the form.
Usually modified by dragging the field.

Y-pos

10

The current value of Y-position of the field on the form.
Usually modified by dragging the field.

Label

15

The label to appear for the field on screen.

Refer

20 The "refer" label used to access the field when building
the initial data dictionary

Display (Type = Text only)

25 The number of characters to be displayed on screen. 0
(zero) for default.

Size

Functions

30

For information on defining functions in your application
see -

Functions Menu

35

see also Function Primitives

For Describing any function within the "target" to the
system, or in program terms for writing scripts see

5

General Purpose Functions

Use this selection for describing functions

10 Label

The function name

Description

15

A brief description of the function. The function can be
located by description

Action

20

A window to the function actions. Double click on this
window to see or edit the full Function Action. see also
Function Primitives & Function Primitive Categories. see

25 Function Action

Double clicking on a function action block brings a panel
into view for editing the function actions.

30 Adding Actions

Select the appropriate action from the alphabetical list
beside the add option, select add and then click on the
panel at the appropriate position. Clicking over an
35 existing action will insert the new action before the
existing action

Deleting, Actions

- 5 Select delete and click on the action. Take care to
 deselect delete before clicking on other actions.

Editing Actions

- 10 Double Click on any action to activate the edit dialog box.

Function Index

Shown for reference purposes. Cannot be changed.

- 15 Strings

See your driver information. Currently this information is
not used by most drivers.

- 20 Function Activation

Specifies when this function will be executed. see

Starting A Function

- 25 Function Number

Each function may be assigned a number. The operator may
then enter the number and the program easily select from
the list using the Function# primitive.

- 30

Hot Key Code

- 35 Each function may be assigned a hot key code. Enter a non-
 zero code in HEX and if a key with this code is pressed at
 idle, or any other time hot keys are activated, the
 function will be activated. Note that the Cancel Key is

regarded as system event.

System Events

5 System Events are similar to hot keys, only instead of keys
being pressed (Note that the Cancel key, IS a system
event), other actions on the target device are involved.
For each target machine a list of System events is
maintained, but these should always include the standard
10 events. Only one function may be assigned to any System
Event.

See

Standard Event Codes

15 Keyboard.

Keys on the keyboard with a value less than 128 (0x80
hexadecimal) generate an event code with the value of the
key.

20

Other event Codes

0 =Reserved

1 System becomes Idle

25 2Cancel Key Pressed

3System Power On

4Numeric Entry

5Smart Card Insertion

6Smart Card Extraction

30 7Magnetic Card Swiped

8Checksum Error Detected on Batch

9Checksum Error Detected on Data

Card Set

35

Select a card set. When any card belonging to this set is swiped at idle, the function will be activated.

Usage Flags

5

Operator Function - The operator of the target device selects the function

Library Function - The function is an internal "subroutine"

Not Used- The function is not used

10 For adding actions to functions which may be varied by issuer or by acquirer see

Transaction Function Input

15 Transaction Function Approval

Function Primitive Categories.

20 Function script is a sequence of calls to system and user primitives. For information of primitives available see

Function Primitives

25 Assignment Primitive

Field1 := String/FfeW

Set field1 to the string/field2.

30

=> (goes to) primitive

=> field

35 The value of the last logical or other operation using the "calculation result" is stored in field

specified

eg

Account == 000

=> zAccount

5

Would set the field zAccount to 1 if Account was zero,
other size zAccount would be zero.

=> result (goes to) primitive

10

=> field

The value of the last logical or other operation using the
"calculation result" is stored in field

15

specified

eg

Account == 000

=> zAccount

20

Would set the field zAccount to 1 if Account was zero,
other size zAccount would be zero.

see also

25

Calculation Result.

Calculations generate a "Calculation Result". Think of
this value as the value you would see on the display of a
calculator if the calculation was performed on a
calculator.

30

Math Primitives

Field1 += Number/Field2

35

Field1 -=Number/Field2

Field1 *=Number/Field2

Field1 /=Number/Field2

Field1 is modified by thefield2/number.

5

Relational Primitives

Field1value1 < Field/value2

Field1value1 > Field/value2

10 Field1value1 == Field/value2

The two fields or values are compared. If one field is text and the other numeric then the return value will always be false.

15

<> != >= <=

For not equals (whether thought of as<> or!=), greater than or equals (>=) and less than or equals (<=) use the opposite case. With the WHILE PRIMITIVE and REPEAT UNTIL primitive then use the NOT option. With the IF PRIMITIVE use the ELSE clause.

20

AMENDED SHEET (Article 34) (IPEA/AU)

Abort Primitive

25

abort

This primitive causes the target device to stop all functions and become idle

30

Alarm Primitive

Alarm (noise type)

35 Makes the sound specified

1 error alarm
2bip type noise
3most severe alarm

5 Bit Manipulation
Bit Numbering

All binary fields can be accessed as a number of bits where
the number of bits $\text{no.of.bytes} \times 8$

10

The MSB of each byte has the highest bit number and the LSB
the lowest bit number. Note ISO bitmaps do NOT follow this
rule, but these do not need bit manipulation by the
application.

15

This result in the LSB of the last byte being bit 0 (zero)
and the MSB of the first byte being $\text{no.of.bytes} \times 8 - 1$. Eg
for two bytes 15.

20 Bitcount Primitive

`bitcount(field,start,end,bitvalue)`

start & end

25

These are both bit numbers. see bitnumbering. Direction of
counting is from start to end, either may be the larger
number.

30 bitvalue

0 indicates count zeros

1 Indicates count ones

2 Indicates counts zeros and stop at the first non zero bit

35 3 Indicates count ones and stop at the first bit no set to
one.

Notes

The number of sequential bits of the value "bitvalue"
5 starting from bit "start" and working towards "end" is
counted.

If the result is non-zero the logical true status is set,
otherwise the logical false value is set, allowing "if'type
10 tests of the result

The count is stored as the "working value" allowing storage
via the "->" (goes to) primitive. see -> (goes to)
primitive

15 Setbits Primitive

Setbits(field,startbit,endbit,value)

Bits number "start bit" thru "endbit" are set to the value
20 "value". No all values are extracted from the least
significant bits of value. e.g. For a 1 bit field, all
values are considered either 1 or 0. (Odd numbers are 1)

Batch Primitive

25

Batch(Operation)

Operation 0 = reset to first txn in batch

Operation 1 = find & restore next txn

30 Operation 2 = delete current transaction

Operation 3 = delete all transactions

CardRead Primitive

35 Card (string, field form, default)

This primitive is identical in operation to the show primitive, with three extra facilities

5 1Input is terminated by either the introduction of a smart card, or the swiping of a magnetic card.

2Data from a magnetic card read is stored in the reserved fields

10

3If the (card entry mode) is non zero, this primitive does nothing. This allows logic to read a card only if the card is not already read.

15 See also

Example

20 A function requiring input of both "Tip Amount" and "Cash Out Amount" can input both using the same field.

Create a form as follows. Edit the PFIELD to indicate an input field.

25 PSTRING Name: Form
PFIELD

Create a function

30 Show(Tip, TipAmt, Form, 0)
Show(Cash, CashAmt, Form, 0)

35 Where "Tip" and "Cash" are strings. On the first call to show the display will prompt "Tip" and accept input into the TipAmt field. On the second call to Show the display

will prompt "Cash" and accept input into the CashAmt field.

Show (string, field jorm, default)

Action

5

This primitive is specialised for displaying input forms. Two parameters (string and field) substitute with PSTRING and PFIELD in forms, allowing the same form to be used for multiple inputs.

10

String

String to replace the PSTRING field on the form. <none> if unused.

15

Field

Field to replace the PFIELD field on the form.

20

Form

The Form may be selected from the list box- or alternatively by selecting Field->Value[X] taken from a field in the data base.

25

Default

A value of one (1) if the existing value of the field is to be displayed as a default, otherwise 0.

30

Reserved Data Dictionary Settings

The driver in the "target" makes direct use of some fields in the data dictionary. Using these table#/field# settings for other use will have strange results and is not recommended.

35

Table 1 (Transaction Table)

Fields in this table may be initialised to default values
5 only. The first fields in the transaction table are
reserved for (in order)

1ROCNUM
2Track 2
10 3Track 1
4Track 3
5PAN
6Expiry Date
7Customer Name
15 8Protocol Status
9Card Entry Mode

Table 2 (Totals Table)

20 No reserved settings, however a fixed ten copies are
available. Initialisation of fields to default values
only.

Table 3 (Terminal Table)

25

This table is the basis of the build of terminal groups,
and may be initialised from the Initial Data Table. There
is always only one record in the table.

30 Table 4 (Issuer Table)

One record per issuer, with the current record selected
automatically when a card is swiped.

35 Table 5 (Acquirer Table)

One record per acquirer, with the current record selected automatically when a card is swiped.

5 Table 6 (Card Table)

Fixed layout, Dictionary specification currently ignored.

Coffind Primitive

10 ColFind(value, LowField, HighField)

Both LowField and HighField must be in the same table.

This table is scanned for a column with the value 'value' between the two fields. The primitive is normally used to
15 locate the CardTable Column for a card. The result variable is set to 0 if no match is found, or 1 if a match is found.

ColSelect Primitive

20 ColSelect(Column , Table , Reset)

Selects the relevant column of the table indicated.

Column

25

The column to use. The transaction table has only one column, the totals table has ten. The other tables (issuers, acquirers etc have one column for each record in the corresponding initialisation data base

30

Table

For comparability, 0 (zero) selects the totals table. The tables are as follows

35

- 1 Transaction
- 2 Totals
- 3 Issuers
- 4 Acquirers
- 5 5 Card Ranges

Reset

If reset is 1 then all fields in the column are reset.

10

CommStat Primitive

CommState(port, value, field)

port indicates the port to be tested

- 15 value indicates a value to compare and set the status accordingly.

field (optional) indicates a field to store ComsState Value.

- 20 This function reads the state of the port store the value in field (if specified) and sets the current function result status to true if the value matches "value".

Date Primitive

Date(commndnd , date-field , time-field)

25

1 Read system date into date field and time field

2 Set system date from date field and time field

see also

- 30 Dates and Times

Dates & Times are special data types both are stored as special numbers.

- 35 Date Fields

A Date field is a two byte number, representing a date since 1Jan1940 to 1Jan2110. Subtracting two dates reveals the number of days between the dates, dividing by 7 reveals the day of the week (Monday =0, Tuesday = 1 etc).

When moving to or from a text field a date is converted to a text format of DDMMYY. If a format is used this may be converted to DD/MM/YY by using a currency symbol of / in the format. The text format of a date may be either 6 or 8 bytes long- showing the year as 2 or 4 digits.

Date Fields only contain valid dates. Since every date is stored as a day number, the storing the string 32/01/1980 will give is the actual date 01 /01 /1 980. If data is entered directly into date fields, then dates are corrected in this manner automatically. If you wish to check the was entered correctly, then enter the data to a text field, then move the value to the date and check it is equal to the string.

e.g

Repeat

Print(GetDdte,0)

ActualDate := TextDate

Until ActualDate == TextDate

The above example will continue to ask for a date until a valid date is entered.

Time Fields

Time fields may be either two or three bytes representing either the time of day to 2 seconds (two bytes) or 1 /1 00 of a second (three bytes) accuracy.

Moving a time to a 2 byte integer gives the number of two second periods elapsed this day. Moving to a 1 byte integer extracts the 1 /1 00s second fraction (up to 199)

Moving to a value or larger integer extracts the total number of tics (1 /1 00s sec) which have occurred prior to the time.

5

Moving a time to of from a text field results in either HHMIVISSFF when moving to a 8 or more byte field and HHMMSS when moving to/from a six byte field.

10 This text value may be formatted with a format to give HH:MM:SS.FF

Moving values between data and time fields and other numeric types results occurs without conversion. Moving to
15 and from text values results in conversion. See specific entries for conversion details

Dial Primitive

Dial(phone numberphone number)

20

The numbers specified must be fields. Immediately following each number field in the data dictionary must be a timeout field then a retry field and then a mode field. The upstream prot is implied.

25

Do Primitive

Do(Function)

30 also known as

DoFunc(Function)

This primitive is used to activate another script function
35 as a subroutine call

Event Primitive

Event(Function, system event)

- 5 Sets the specified function to be activated whenever the event occurs

Exit Primitive

Exit(Now?, Value)

10

This primitive is used to set the return value of the current function, and optionally, exit immediately.

- 15 The 'Value' is stored in the Calculation Result, which will be regarded by any calling function as a result.

If 'Now' is true (is 1) exit will be immediate, otherwise the exit value will be established.

- 20 Func Number Primitive

Function#ffieldlnumber . bad number function)

Execute the function with the assigned function#.

- 25 Typically this primitive will be used by a user function set to be activated by a Hot Key on the target device labelled "Fn" or "Function" or similar. Such a user function would prompt for a number and then call this primitive (Function#) with that number as a parameter.

- 30 See Function Activation.

The "Bad - Number - Function" is a function in the script to be executed if the no function matching the first parameter exists.

35

This is used to implement number functions- for example clearing memory might be function 1055. The user presses the "Function" hot key, then enters 1055 to execute the function.

5

To achieve this

1a function containing this primitive should created and set up under function activation to have the appropriate key code.

10

2A function containing the appropriate action for the numbered function should be created and set up under function activation to have the appropriate function number

15

The function number also returns the logical result of the request to call the numbered function. i.e false if no function exists, otherwise true.

20 If Else End Primitives

If

The next primitive is executed. If true then all primitives between the if and the else are executed. If false all primitives between the Else and End are executed. If nothing is required between for false then Else may be omitted.

30 If ! (if with <not?> parameter)

Else

Optional in an If see above.

35

End

Marks the end of an If or While. See While End

5 KeyBd Primitive

KeyBd(mode) - (O = off / I = on)

10 Upon entering idle state, the action of the keyboard is
determined by the last KeyBd command. The keyboard (except
cancel) will be ignored if off was specified.

MAC Primitive

mac(key,mode,message,field)

15

All targets must support the storing and use of 4 64 bit
keys.

mode I

20

Calculates a mac of the 'message' and stores the value in
the 'field' specified. Uses the 'key' specified. If
the 'message' is 8 bytes in length only (or less) then a
single DES encryption only results.

25

mode 2

Stores the specified key into secure memory from 'field'

30

Mod

Mod(Value,Divisor)

The value "value" is divided by the divisor, and the
remainder is the result.

35

e.g.- the following example would set the Data Field
"Remainder" to 4. (25 divided by 7 has a remainder of 4.

Mod(25,7)

5 => Remainder

Pin Primitive

pin (field)

10

Retrieves pin block from pinpad. Not supported on all
CardScript devices

Print(Display/Report, Part)

15

Form

The Display/Report may be selected from the list box- or
alternatively by selecting Field->Value[X] taken from a
field in the data base.

20

Part

Values - 0 = all, 1 = pre print/header, 2 = body, 3 = post
print/footer

25

see Forms -end of Header/PrePrintt & Start of Footer/Post
Print

Action

30

The selected section (or all) of the display is displayed
or, in the case of a report, printed.

With displays, any input fields will be accepted, however
the Show Primitive is recommended for input operations

35

ProcDown Primitive

Procdown(protocol, port)

- 5 The specified protocol is started on the port specified.
This function is normally used for downstream protocols
such as an ECR. This function is intended for more
advanced users and the protocol must specify its own
success and fail functions.

10

Protocol Primitive

Prot (protocol, Function 1, Function 2)

- 15 The specified protocol is started on the bank coms
(upstream) port. The current function execution continues.
KeyBd(Off) is set (it may be overridden). If the
protocol returns a value of zero, Function 1 will execute,
if any other value is return Function 2 will execute. (A
20 KeyBd(On) will automatically happen before either function.

Range Primitive

Range(field,Min,Max)

- 25 Returns true if the value specified is \geq min and \leq max
value

Repeat /Until Primitives

- 30 Repeat

Repeat sets the execution point for a following until

Until(Case) Cases are 0 = False, 1= True

35

Executes the next primitive and if the result agrees with Case then the Repeats everything after the repeat primitive.

- 5 Report (Form, Function)
 Action

First prints any pre print or header from "Form". Then for each transaction in the batch calls "Function" and prints
10 the details section of "Form". After cycling throughout the batch, then prints any post print data from "Form".

Form

- 15 The Display/Report may be selected from the list box- or alternatively by selecting Field->Value[X] taken from a field in the data base.

Function

- 20 A function to be executed before each detail section is printed. For any transaction in the batch for which the function returns FALSE will be skipped.

- 25 Restore Primitive
 Restore(layout,field,secondary field)

This primitive is used to retrieve information from the Batch Area.

- 30 Layout

This optional ("none" is permitted) parameter specifies which transaction layouts are considered for retrieval.

35

WARNING! All records searched using a field other than RECNUM are actually retrieved, changing the contents of the data fields in their layout. Using a value of "none" may have side effects!

5

Field

The primary search field. Searching will advance through the Batch Area until a match is found.

10

Secondary Field

an optional (Use RECNUM for "none") secondary search field.

15

Rom Function Primitive

Rom(valuelfield, Message)

Generally the parameter passed is passed directly though to the bios. The following values are assigned for portability. The message parameter is ignored unless otherwise stated.

20

1Go to ROM mode.

25

2Erase memory and return to Rom

3Start TIVIS download (from ROM mode).

4Store Rom Params.

30

Uses Message and returns Success status.
See ROM SETTINGS.

51-load Rom Params.

Uses Message and returns Success status.

35

See ROM SETTINGS.

6Activate Rom Edit.

Returns Success status. See ROM SETTINGS
see Also

5

ROM SETTINGS

What are ROM SETTINGS

Sub Heading

10 Normally target devices store programs in RAM memory, and
are capable of loading these programs over the telephone
Network. In order to achieve remote loading the device
must store telephone number and other details required.
15 The device must have a method of loading and/or editing
these details.

Methods vary from device to device with information
normally being obtained from the keyboard, a smart or
magnetic card or some combination. Obviously the
20 information must be able to be set prior to the application
loading.

Communication Between Rom & CardScript

25 Two possible reasons for CardScript to interact with the
ROM Settings arise.

1 The parameters may need to be changed in a device already
loaded with the CardScript application.

30

2A CardScript application may need access to the ROM
Setting values.

Edit Rom Settings - Rom Function 6.

35

The desired method of allowing change to the settings is to use this function primitive call. (see Rom Function Primitive.) This primitive may not be supported by all Drivers and (check with the driver provider) but provides
5 the only device independent method of implementing the function. An advantage of this function is the operator sees the same interface as when configuring the terminal prior to loading CardScript.

10 Load Rom Settings -Rom Function 4.

This function is used to obtain the ROM settings in a Script.

15 Store Rom Settings -Rom Function 5.

A Script Program may load the Rom Settings with Function 4, allow editing of values and use this function to store the settings. It is recommended to use function 6 (edit) in
20 place of this procedure where available as this mechanism allow changing of device specific settings.

The Rom Communications Buffer.

25 To provide a much device independence as possible using functions 4 and 5 CardScript defines a standard Communications Buffer Layout with a private area at the end. All Fields are ASCII.

30 The first three fields are assumed to be used for all communications. 2 bytes connection mode. Lan Leased Line etc 4 bytes station/Lan Address
8 bytes telephone prefix - eg "9,"
Field 1 16 Bytes Terminal ID. The ID as seen by the
35 software management system and not

necessarily other systems.

8 bytes terminal type

5

24 bytes phone number

24 bytes connection string

10 Save Primitive

Save(transaction layout)

Saves the current transaction to the batch using the layout specified. A new Transaction Index is generated according to the method specified by the last TxnIdx: Primitive, with the new index stored in field(0,0) RECNUM. For details on RECNUM see Reserved Data Dictionary Settings.

The number of transactions (of the selected layout) which can be stored is returned. If zero is returned, then the save could not work! If 1 (one) is returned then no more may be saved. If two is returned then only one more may be saved, etc.

25 Store Primitive

Store(offset,messageLayout)

The store Primitive stores the last received message, starting at byte <offset>, using the specified message layout. The function result status is set by the operation. (Set to FALSE if the store did not match).

Tots Primitive

35 Tots(valuer Field)

Selects the relevant totals column.

This primitive has been replaced by the ColSelect
5 primitive. Old programs are automatically upgraded, since
parameter 2 or LineTble, when zero, will select the totals
table

TxnIdx Primitive. Set Transaction Index.

10

TxnIdx(Field1,Field2,mode)

Field1 is optional. If include the first two digits of the
Index are set from this field.

15

Field2 specifies the main field on which the Index is
based. By default this is the ROC field.

the mode specifies how CardScript increments the Txn Idx.

20

0=add 1

1=Amex Style

2=None. Incremented by script.

This function would normally only ever be used in a start
25 up function. The calculated value is always stored in the
ROC field.

User Function Primitive

30 The user function primitive is used to call any of a range
of functions. The functions call by user function are NOT
standard.

Primitive - Extensions

35

It is possible to extend the primitives available to CardScript. The extensions take the form of a block of 'C' code loaded with the Script. 'C' code, of course, has the restriction of being non portable.

5

The existence of these extensions is to allow extensions to a set of primitives to be tested without changing the core driver. Any extensions initially tested by this means must be added to the set of primitives in a new release,
10 otherwise the code calling them will never be portable.

Wait Primitive

wait(minutes, 100msecs)
15

The current function pauses for then number of minutes + 10ths of seconds specified. A delay of up to 1000 minutes (over sixteen hours is possible) and as small as 1/10 of a second.

20

While / End Primitives

While(Case) Cases are 0 = False, 1= True

25 The next primitive is executed. If the result matches Case then all primitives between the While and the End are executed, then we come back again to the While. If the result does not match Case then execution continues with the primitive following the End.

30

End

Marks the end of an If or While. See also -- If End
For specific categories of primitives see

35

Communications Primitives
Data Entry Primitives
Displaying and Printing

- 5 For information on configuring CardScript for target device.
function primitives (advanced users only) see

Configure Function Primitives

- 10 For advanced users only!!

Usage - Name Changes

- 15 Changing the name of a primitive or a primitive parameter
will cause all scripts using the name change to be
automatically updated. Both this type of change and any
changes to the "infix" status of a function will have no
affect on the driver and scripts will function without
further change.

- 20 Usage -Adding, Deleting, Changing Parameter Types

Parameter Settings

- 25 Each function parameter has the following possible
categories
1A Field from the data Dictionary
2Numeric Value -Which may be displayed as an index to a
file
30 3A String
Any parameter may legally accept any combination of
categories

- Layouts
35

CardScript allows you to graphically enter your layout specifications. For details on Receipts, 'Reports, Displays, Messages, Protocols, and Transactions see Layouts Menu

5

Layouts are the main engine of any application. Although all layouts must be brought into operation by functions, layouts also in turn launch functions and other layouts.

10 Form layouts, message layouts, and transaction layouts are similar in operation. All three are an arrangement of fields and strings called a field panel. For details see

Field Panels

15

All field Panels (Displays/receipts, messages and transactions) have a Panel Control box in common. The selection in the Panel Control box selects the action to take place when the left mouse button is clicked over the panel.

20

Additional controls are present of some panels, however, this box always contains

25 An add field -control with field edit box and palette selector

Clicking on the p'a'nel when [add] is selected adds a new field as displayed in the edit box

30

Before - clicking on the edit box to set the field to be added, select the appropriate type of item in the "from palette" drop down list box

35 Clicking on the edit box brings up either the Select Field Dialog or the Enter String Dialog, in accordance with the

palette selector

A dealt field control

- 5 Select [delete] and then click on the appropriate field

A select field Control

- 10 Select [delete] and then click on the appropriate field
Field Editing

To edit any field, double click on the field

- 15 Forms (Displays and Reports)

see also Field Panel, Print Primitive, Show Primitive and
Report Primitive

- 20 The Screen is Divided into four sections

The Form (Display/Report) Panel

- 25 The panel is a Field Panel where the location of the of
each field corresponds to the place
actual data will appear on the display/printer

The Dashed Boundary

- 30 Depending on the Display/Report type, a dashed line will
appear showing the limits of the display or printer. This
boundary is drawn in accordance with the settings in the
Tables/System menu and can be changed at any time.

Form Name

- 35 The display report name is used for reference to this
screen and should contain a

meaningful name.

Panel Control

- 5 In addition to the panel controls discussed in Field Panel, two additional controls are present.

<<End of Pre Print

- 10 Pre-Print fields appear with a grey background
Select this item and click on the field after the end of the header section of the report.

- 15 Used in reports , the header section is printed once at the beginning of the report. The sections following the header will be printed once for each transaction in the batch. see Report Function for further details

- 20 Used in receipts (see Print Function for further details)
used to divide the receipt not sections

Start of post print

- 25 Post-Print fields appear with a grey background, and can only be distinguished from Pre-Print fields if there are fields in between. (As would normally be the case.)

- 30 Select this item and click on the first field of the post print section of the report.

- Used in reports , the Post-Print section is printed once at the end of the report. see Report Function for further details

- 35 Display/Report Type

The types are

5 Display - layout will always appear on the display, and in
scribe will have a border reflecting display width and
number of lines

Secure Display - reserved for future use

10 Printout - layout will always appear on the printer, and in
scribe will have a border reflecting printer width

Soft Keys

15 The Soft Keys Button allows selection of a soft key set.
see

Messages

20 Output messages

A messages buffer is built from fields in the data
dictionary, and from strings in much the same way a
printout is build. However, in messages, all data may be
25 represented in forms other than ASCII. (see "the message
engine". Formats may be used to specify data within the
selected representation.

Input Messages

30

Messages are also used to specify how data is transferred
from a received buffer and stored in data fields.
see also

35 The Message Engine (Processor)

The message engine is used both to transfer information both from data fields not a message buffer, and from a message buffer to data fields

5

see also

Message Data Mapping

10 Every field in a message buffer is converted from the type in the data field, to the representation of in the message buffer. For "Forms" all data in the buffer is Ascii, but in other messages the data may be any of the following

15 Ascii

Ascii Representation

Strings and Text Fields

20

Strings & text fields are simply copied. If the source is shorter then the destination, spaces are used for padding

Integer

25

Integer to Asch

For one & two byte integers, the binary value is converted to its string equivalent and then formatted according to
30 any format specified. Larger integer conversion may appear in a later release

Ascii to Integer

35 Again limited to 1 and two byte integers, the value of the text is calculated and stored in the integer.

Amount

As for integer.

5

Dates

Dates are converted to either DDMMYY or DDMMYYYY if the
Ascii field is longer than 7. Formatting is applied on
10 conversion to ASCII only.

Times

Times are converted to either HHMMSS or HHMMSSFF (where FF
15 is the fractions of a second in hundredths) if the Ascii
field is longer than 7. Formatting is applied on conversion
to ASCII only.

Hex

20

Hex Representation

Amounts

25 To Hex: The data is converted to a BCD string and then
expanded

Integers

30 The binary value is converted directly to Hex. eg a one
byte value set to 35decimal (23 hex) would be converted to
two bytes - characters' (0x32) and '3' (0x33) representing
the hex value 23.

35 Binary

Binary Representation Integer

- 5 Binary representation of integers is High Byte Low
Byte. As a binary value. No Actual conversion takes place

Text

- 10 Binary representation of Text is to assume the text is a
hexadecimal string and convert this to binary. To get an
exact copy of the string use Text Representation.

BCD

15

BCD Representation.

The Data is converted to the BCD data type.

- 20 Formats

Formats are used for specifying exactly how data will be
represented

- 25 Justification

Any time the data length is less than the field width, the
justification will be used to decide where the data is
placed.

30

Allowed Characters

Specifies the type of characters allowed during input.

- 35 Minimum Characters

Specifies the minimum characters allowed for entry to a field.

5 Maximum Characters

Specifies the maximum characters allowed for data entry, and the maximum displayed characters on output.

10 Input Window

A non-zero value in this field specifies input will occur within a window. e.g A 24 character text field may be input using a 10 character window because of limited display space. Note that only ten characters of the input would then be visible at any one time.

Input Validation

20 A function may be specified here to valid input using this format. The validation function may store the current input using the ->(res) primitive. See also the Console Primitive command 4.

25 Note that an input validation function MUST NOT do any displays, as the current display would be overwritten.

Suppress leading zeros

30 Check here to suppress leading zeros in numeric fields, or leading spaces in text fields

Decimal places

35 Select the appropriate number of places, and the character to use as the decimal indicator. Selecting the decimal

(from '.' or ',') also determines the character for thousands separation. (The opposite character to the decimal is used for thousands.

- 5 Check the box to require keying of the decimal indicator during input. If this box is left unchecked, data 1. 00 (. as decimal) would be input as 1 00, cash register style.

Auto OK

10

If this box is checked, when the maximum characters are entered, input will be concluded.

Thousands

15

The thousands separator (as determined under decimal places) will be automatically inserted.

Password Mode

20

The first character of the currency symbol will be displayed in each position, in place of the actual character entered.

25 Currency Symbol

- Specify if the currency symbol is to be displayed. One or two characters may be entered. If the value is two digits, the digits are legal hex digits then these will specify a the character. e.g. 41 would specify the characters', as would a single A character.
- 30

This field as has other uses.

- 35 The separator for date & time fields is the first character. For time fields the second character is used to

separator the hundredths of seconds when displayed

Length Indicator

- 5 In addition to the data, the data length is to be included.
The number of digits to use may be 1,2 or 3. The length may
be before (pre) the data or trailing (post). As an
alternative to a numeric length specification the currency
symbol may be used to indicate the end of a variable length
10 field

e.g.

length as 1

- 5abcde is a five character field

length as 2

- 15 - 05abcde is the same field
currency symbol as ':' and use currency selected
abcde: is the same field again
Pad BCF with F

- 20 Check here for BCD fields of odd length to be filled with a
trailing F nibble. If unchecked a leading zero nibble
would be used.

Transactions

25

Two other layouts types are also available

Bitmaps

- 30 Protocols

- Protocols describe message flow both from and to the target
device. The top line specifies outgoing messages and the
other lines display possible incoming results. A protocol
35 consists of lines and sections.

Request Line

- At the start of each section is a line 1 (optional for the first section) which describes the outgoing message. This is the request line.

Response Lines

- Lines 2, 3 and above define actions to be taken when a response is received. These are the response lines. A response may be a data received or a time out. When a timeout occurs the first line with a timeout will be selected, any other line with a timeout will never be used. When data is received, all lines beginning with a message are tested to see if received message matches the requirements. The first item on each response line must be either a message or a timeout.
- Protocol Screen Editing

Adding Entries

- Select the desired item to add, then click on the display at the desired location

Splitting Sections

- A section may be split on line 1. Click below the line slightly to the left of the field to become part of the second section.

Adding to a Line

- When adding fields click on the field that will be after the new field. To add to the end of a line click about 3

spaces beyond the end of the line. Always click on the desired line.

Inserting a line.

5

Click where the new line should start

Retrys/Skips

- 10 After entering a Retry, the retry field will be selected. Or you can select the retry later. Once a retry is selected the <<set retry>> and <<set skip>> commands can be used to set the points where execution should move in the event of either a retry or the retry count being exceeded.
- 15 Note, retry can only move back and skip can only move forward... For those with colour displays, the retry arrow is green and the skip arrow is drawn as red. You can't put a retry/skip on line one.

- 20 Identifying Input Messages

The first field of each input line is used to select when the input is appropriate. The following possibilities are catered for

25

Control

The input line is selected when the a message begins with the specified character

30

Message

The incoming message is matched against the message specified.

35

Timeout

A timeout line will automatically be selected in response to an incoming timeout.

5

Function

If the function returns true the message is matched. The Store Function.

10

Functions Launched By Protocols

Within protocols it is possible to launch functions for various reasons, particularly to store complex messages and select options.

15

Such functions should NOT halt operation, either by WAIT(or for input or any other event. Should a function attempt to do so, subsequent functions launched from the protocol, including the "good" and "bad" functions, will execute before the function resumes.

20

Delay any inputs until the good or bad functions at protocol end. If you are an expert user and must do an input, make it the last command in the function and exercise caution.

25

Repeated Messages

A protocol may involve repeated messages. That is, after storing the data from an input message, another similar message will be received.

30

Fixed number of repeat messages

35

If the number of times a message is to be received is fixed then the following approach may be used

<Msg><Retry(nn)>

5

Where nn is then number of messages expected

Variable number of repeat messages

10 If the number of times the message will be repeated will vary, i.e a flag in the message indicates that a repeat message will follow, then the following technique is recommended.

15 use an input line with <Function><Retry(0)>

This will cause a loop whilst the function returns true. From the function, use the store primitive to save the data and return true if another message is expected

20

Layout Primitives

Layouts use the following elements as building blocks

25 Putting it all together

Build Menu

Build Target Group Files

30

Builds the font and conf files ready for program execution

Build Script as Fragment

35 Builds a reduced script for loading either onto a smart or through the communications network, for describing a

particular operation which may be changed without loading a new program.

Build Secure Prompts

5

Builds the list of secure displays and associated strings for loading into a secure display.

Run Simulation of Build

10

Activates the terminal simulator program

see also

15 Files Produced By Build

Font Files

xxx is the number of the font. Currently always zero

20

fontlxxx.bl1

Characters 0.. 127. Bytes are dots across. First byte is top row. If more than 8 dots across, then the next byte continues the dots

25

fonthxxx.bl1

Characters 128.255, using the same format as theTfile

30

fontdxxx.bl1

Characters 0.. 1 28. Bytes are up and down. First dot is top left (bit 0 of byte 1) then dots down the character.

35

fontuxxx.bll

Script Fragments

- 5 Script fragments are small scripts (usually < 256 bytes) built separately to a main program. These scripts may then be loaded into the terminal (either from a smart card or as part of a message) in order to specify operation of changeable program feature

10

Examples of Fragments

A Fragment for User Authentication

- 15 A Smart Card could contain a program fragment specifying how the terminal should check the user of the card is the real owner. Then cards may be issued with varied scripts such as

Input & Check PIN

20

Print A Slip and request a signature Do nothing- no check

A fragment for communications protocol

- 25 A server could have a list of communications protocols of various networks. Then the terminal could dial the server and request the relevant fragment for a particular network (either because the terminal has no information on the network - or the existing protocol no longer functions), allowing the terminal to operate on a new or
- 30 changed network with obtaining a complete new application.

Menus

- 35 File Menu

Configure Menu

The Configure menu is greyed on standard level Scribe. The functions available are for the use of advanced users only.

- 5 Generally within an Organisation using CardScript either one master user will be placed in charge of setting configurations or configurations will be set by an external consultant.

- 10 The configuration options are

Configure Simulation

Host Comms

15

Select a comm port for the simulator to use for modem communications. If none are available select "none". Selecting "none" precludes testing comms facilities

- 20 Terminal Group

The Build process creates several build files one for each terminal group. The setting chosen here determines which build file will be used for simulation.

25

Cards

The simulator does not use a real card reader. Instead it supplies card data from this table. Enter card data as required for up to eight test cards.

30

To provide the simulator with information on this PC and to create test "Cards".

Configure Targets

35

A number of target machines may be described to the CardScript system. The information about the target machines is used in various places throughout the scribe system to present information in a manner appropriate to a currently selected "target" machine. see System Record information for setting a Current target.

Object types

10 The target machine is described by arranging an number of "objects" on this panel. Their is one of each of the display, printer, and reader objects, and as many button objects as are appropriate.

15 The display object is used to specify the display configuration in lines and columns. This object should be dragged to an appropriate location in the window.

The printer object is used to specify the print width columns. This object should be dragged to an appropriate location in the window. The number of lines setting bears no relation to the number of lines on a printer page, this field determines how many lines will be available for viewing during simulation.

25 The reader object is used to describe which area of the window will be used to display buttons for simulation of a card reader. This area bears no relationship than actual card reader. Draq this object to an otherwise unused area of the window.

Any number of button objects. The object correspond to the push buttons on the target device. Five different button styles are available. Configure these styles as required.

35 When a style is changed, all buttons of that style will change in appearance. Keycodes returned should match those

returned by the actual terminal B10S. Use the KeVMap Primitive to force the map these codes to the codes required by the actual application.

- 5 For describing the various hardware platforms to the system

For designing Tables Screen Layouts and Contents used on the PC with Scribe

- 10 For designing the Data Dictionary used in the Target Device

For Specifying the functions available within the target device.

- 15 Reference

Index
Glossary

- 20 #

"->"(goes to) primitive: <goes to primitive>

"KeyBd": <KeyBd Primitive>

"refer":<ColSelect Primitive>

"target": the PC, EFTPOS terminal , PiNpad or cash register

- 25 which will be used to run the
developed application.

->(res): <result goes to primitive>

B

Batch Area: Storage area of memory. Used for storing

- 30 transactions and any other miscellaneous data. Also may be
thought of as file storage.

bitnumbering: <Bit Numbering>

C

- 35 CardEntryMode: <Card Entry Mode>

ColSelect primitive:

Commands for Memory Cards: <Commands For Memory Cards>

Console Primitive: <Console Primitive>

D

5 Data Dictionary Field Attributes: <Data Dictionary Field
Attributes>

Date & Time Fields: <Dates and Times>

E

eserved data dictionary fields:

10 F

Field Panel: <Field Panels>

Forms -end of Header/PrePrint & Start of Footer/Post
Print: <Forms>

Function Action: <Function Action>

15 Function Actions: <Function Actions.>

Function Primitive Categories: <Function Primitive
Categories.>

Function Primitives: <Function Primitives>

H

20 HEX: . Digits 0..⁹ and A..F

I

Initial Data Field Attributes.: <Initial Data Field
Attributes>

K

25 KeyMap Primitive: <KeyMap Primitve>

P

PFIELD: special field for use on Forms. In place of this
field a supplied parameter field will
be displayed

30 Print Primitive: <Print Primitive>

PSTRING: special field for use on Forms. In place of this
field a supplied parameter string will be displayed.

R

RAD: Application Development (especially used

35 with'tool')The process of defining a program in a very
short time by starting the program definition with the user

interface.

Report Primitive: <Report Primitive>

Reserved Data Dictionary Settings:

Reserved Data Dictionary Settings: <Reserved Data

5 Dictionary Settings>

ROM SETTINGS: <ROM SETTINGS>

S

Show Primitive: <Show Primitive>

Smart Card Type Code: <Smart Card Type Codes>

10 System Table Settings: <System Table Settings>

T

Txnldx Primitive: <Txnldx Primitive>

W

15 Windows: popular operating system for PCs. based on an
event driven architecture.

APPENDIX B

Bios Objectives

20

The objective of the CardSoft BIOS is to make all devices
used for running Point of Sale software compatible with
CardScript programs.

25 Bios Usage

The CardSoft BIOS specification is designed to allow the
creation of portable programs for Payment Terminals. Any
given implementation of the BIOS will encompass its own
30 "look and feel" which, in turn, is imparted to applications
using the system. This is possible since the BIOS
specifies what must be achieved by low level functions,
rather than the manner of achievement. This means that not
all implementations of the BIOS are equivalent and there is
35 scope for vastly different performance and operational
convenience whilst still maintaining BIOS compatibility.

As an example, the BIOS itself does not specify how such things as how cursors and editing functions are implemented, there is simply a call specifying display this field and allow it to be edited. Thus the field editing rules are determined by the individual BIOS implementation. One brand of equipment over type may be standard, on another insert may be the default.

This leaves individual implements with the ability for creativity and a framework which allows for the performance and convenience of their programmers to be a product advantage.

Also supplied in addition to the core BIOS are some implementation routines. These are supplied in source code as a starting point for actual implementation. However the code in these routines is not applicable to all hardware configurations and would expect over time to be modified in any given implementation.

The BIOS described in this manual represent the interface between CardSoft EFT applications (including the driver for CardScript) and an EFT terminal, however the specification is general purpose in nature and may in future be used to support other systems. This manual assumes CardScript is to be supported and is geared to assist in achieving this goal.

In addition to the functionality described here the EFT device must have its own "bootstrap" system. Where CardSoft applications are being added to existing products a software module which interfaces between routines described here and the existing driver software can easily be produced.

This BIOS specification remains the property of CardSoft.

Utilising Existing Operating Systems

When first adding CardScript to a device, some level of BIOS or operating system will normally be already in place.

In many cases it is desirable to add CardScript to devices originally developed years prior with well tested hardware device drivers. In these instances the BIOS will constitute an interface between the existing operating system and the CardScript driver. The BIOS may then be linked with the Driver and the combined application loaded as one conventional application.

The BIOS is designed to be able to be placed as a layer above any pre-existing operating system, and be loaded together with the Driver program as one application to devices installed in the field.

New Products

In the case of new products, created for use with CardScript, a purpose build BIOS will minimise memory requirements and speed time to market.

Steps To Implementation

The steps in implementing the CardSoft BIOS are as follows. Check the BIOS library supplied with this manual is correct for your microprocessor development tools. Other versions of this library for other development environments may be obtained from CardSoft.

Choose appropriate optional code. In order to simplify implementation of the BIOS sample code is supplied for some typical hardware configurations types providing higher level functionality and simplifying installation. It is recommended to make use of this software initially, and replace code as desired once the system is operation on the target hardware. Use supplied outline "main.c" and compile & link.

Add routines to eliminate unresolved externals. Use empty routines supplied for routines to be supplied later. It is recommended to initially include real keyboard and display routines and then add others.

Concepts

Event Driven Structure

5 Applications constructed to run on the CardSoft BIOS must
be event driven. This allows the BIOS or operating system
to have control during idle periods. When an input event
occurs, the application is called to process the input, and
then returns to the BIOS / operating system. The
10 application sets which routine will handle each message and
what messages are enabled.

This event driven structure allows the BIOS to operate as
an interface layer to event driven operating systems
without problems. Where the underlying structure is not
15 event driven This enables the BIOS functionality to be
matched by either low level BIOS code or by a high level
operating systems ensuring maximum portability of CardSoft
application, and enabling sophisticated underlying
structures to be utilised where present. The event driven
20 structure of the system means that applications do not
contain a "main" procedure. Applications have an init-
application(routine which sets up a table of routines
addresses to be called in the case of external events
occurring.

25

Callbacks

Callback Control - Input

The BIOS must maintain a callback address table with four
30 entries for each input/output file. Associated with each
entry in the table is an enable status. When the
corresponding event occurs a callback should be made using
the address from the table. Each callback contains an
optional Code and Message (see below). The BIOS should not
35 issue a second callback while another callback is in
progress. This can lead to race conditions.

Output

For porting the CardSoft Bios to a new machine see

5

Low Level Interface

The low level interface represents the routines that must be custom written when porting CardScript to a new device.

10 These routines assume that the standard Console module, or equivalent are used. Use of these modules eliminates most of the work in implementing the CardSoft Bios, but postpones fine tuning the Bios to make use of Specific hardware in the most efficient manner.

15

Standard Modules

The following modules implement the high level interface
console.c

20

callback.c

math.c

To implement the low level interface, a single module may be created interfacing the foiling routines to the actual hardware or existing drivers. the categories of routine are

25

Low Level Display

void dispbin(uchar ch)

30

Display character ch at current cursor position and advance cursor one place

void cleardisplay()

void dispStr(uchar *str,int len)

35

continuous dispbin for length of str.

- 110 -

length of str is either len characters, or if len = -1,
then str is null terminated.

uchar dispScroll(uchar direction)

5

Directions are

1 left

2right

3up

10 4down

Each call is a request to scroll one place in the specified
direction. The result indicates the success of the request
(1 = OK, 0 = can't do)

15

Low Level Printer

The only printer routine is low level. see

20 Printer

void prch(uchar ch)

This routine simply prints the character "ch" on the
25 printer device.

Special codes are as follows

0xA (10) end of line

0xC (12) end of form. Feed lines as required for tear off
of receipt

30

10 General Routines

uchar softKeyBase(uchar select)

35 By convention returns V for a parameter of 0, and 'a' for a
parameter of 1. Change these to indicate actual key values

for soft key sets

buz

void buz(int freq, int duration)

5

Communications

The following routines must have the code inserted to call the low level drivers correctly.

10

All routines work with a comfile number. Number 0 is the default and is used for the modem. Number 1 should be the auxiliary com port, if present. Number 2 is the second auxiliary (again if present) etc.

15

Sendcom_msg

This routine sends block of characters to the specified port. If low level drivers (such as those used with HDLC) require the block at one time then you will need to call those drivers directly from here. If the target device supports only character mode communications, then the "sendcom" routine may be called once for each character.

25

```
void sendcom-msg(int comfile,uchar *buf,int countOfChars)
{
}
```

Sendcom

30

```
A single character is transferred to the specified port.
/* individual coms character send routine */
void sendcom(int comfile,int ch)
{
35 }
```

Dial

This routine is used to start the dial process. "num1 " is
5 to be dialled "cnt1 times, then if this fails, "num2" is to
be dialled "cnt2" times. "Mode" indicates the
communication mode to be used. These parameters are under
direct control of the application programmer, but by
convention model=async,2=HDLC

10

```
/* MODEM */
```

```
void dial(uchar *num1,uchar len1,uchar cnt1  
  ,uchar *num2,uchar len2,uchar cnt2,uchar mode)
```

```
{  
15 }
```

Hangup

The equivalent of the ATH command on a hayes modem.

20

```
void hangup()
```

```
{  
}
```

```
txstate()
```

25

```
uchar txstate()
```

This return should return a status as follows

0 = busy

30

1Ready

2Reserved for errors, no currently used

Real Time Clock

35 The real time clock is read & set with the biosDateTime()
routine

biosDateTime()

unsigned int biosDateTime(command , buffer)

5

Command (1 =read Dateltime,2 =set date & time from buffer
Buffer DDMMYYYYHHMMSSFF

TPDU - The smart card interface

10

uchar driveTPDU(uchar l1, uchar l2,uchar *Command,uchar
*sendBuf,uchar *receiveBuf)

This routine implements, both the Scribe TPDU and SmartCard
15 primitives. To decide which is call is being made, the
Command parameter must be tested.
Command == NULL, SmartCard Primitive

l1, and l2 are the parameters. Refer Scribe.hlp for
20 details

Command!= NULL - TPDU primitive

This a direct implementation of the scribe TPDU primitive,.
25 with L1 as the length of the sendbuffer, and L2 the length
of the receive buffer.

If L1 is non zero, there is data to send to the card. If
L2 is non ZERO, then data from the card is required.

30

Result

Return zero, unless the function is used as the SmartCard
Primitive, and a result is required.

35

Timer

CardScript requires the target device to have a
100millisecond timer. This timer should may a call to the
5 script routine "time - tick()"

It is recommended not to make a call direct from the
hardware timer interrupt. This would result in actions
launched by time tick() to execute with interrupts off,
10 giving some very strange results.
Instead set a flag in the interrupt handler and have the
event loop clear the flag and call time-tick() (if using an
interrupt handler).

15 The script driver includes the routine "start-bomb()" which
may be called by the bios interface if required

The Font File

20 The font file consists of sets of entries as follows
1 Character code - 1 byte
2Width - 1 byte
31-leight - 1 byte
4Bitmap

25 The bitmap is arranged as follows

For each row of height as many bytes as needed for the bits
(1 for width <= 8, 2 for width <= 16 etc).

30 Left most bit in the MSB of the first byte.

The file is appended with a block of three zero bytes.
(Code, width, Height =0) and no bitmap.

35 For fine tuning an operational Bios see

BIOS Specification (High Level Interface)

By Category Routines are:-

5

Console (Display & Keyboard)

Input

Sequential (Non event driven)

- 10 Since the non event driven machine must be made to appear event driven, the bios interface must include the main line and call the application to handle any events

Sample main()

15

```
{
init - hardware(); /* perform any hardware specific
initialisation */
init - applications(); /* call to routine in module DRIVE
*/
```

20

```
for(;;)
{
    if(event) /* test for event*/
    { clear - event(); /* clear event status */
      handle-event()*/ call CardScript event handle - see
```

25

```
list*/
}
.....
}
}
```

30

Events and Handlers

The following list of events should be catered for

35

Events List

Console

If the high level console is used. Keyboard events are reduced to a single call-Process -

5 Process-key

```
void process-key(uchar key-code)
```

All that is required by the implementor is to map key codes from the actual machine to the those to be seen by the
10 CardScript application.

Please note that the application programmer has the ability to remap the keys using CardScript.

15 Special Key Codes

0xA "Enter" or "OK". (Completion of input)

0x8 "Clr" or backspace

0x1B Cancel

20

The only other console input is the magnetic card reader.
Please use

```
callback(Console, 3, <unused>, buffer, <unused>);
```

25 (use 0 (zero) for unused parameters.

or

```
process-card(<unused>, buffer)
```

Magnetic Card Read Buffer

30

The buffer may include any or all of the following sections. They must appear in order.

Section I (optional)

35 Identifier byte (001)

- 117 -

Track 1 Data - all ascii values

Track2 (optional)

Identifier byte (002)

5 Track 2 Data - all ascii values

Track 3 (optional)

Identifier byte (N03)

10 Track 3 Data - all ascii values

End of data marker

Identifier byte (0x0)

System

15

System Events

For each event, make a call to the routine
void systemEvent(uchar event)

20

Communications

Comms Events

Character comms

25 callback(Port,1,Character,NULL,0)

Message based comms

callback(port,2,0,BufferAddress.MessageLenght)

30 Dial or Tx Finished

When any operation which made the communications port busy
has finished, it should tell the script driver by the
following call

35 callback(port,4,0,NULL,0)

see also

Event Driven Input

Please consult CardSoft for further information

5

Structural

Memory Management

10 The MEMPTR type

External Memory

Often target devices have 8 or 16 bit microprocessors which
can address limited memory without the use of paging. To
15 allow access to such memory, the concept of External Memory
has been defined

It is not assumed that this external memory is directly
addressable by the CPU, instead this memory is accessed
20 only via the memory management functions.

The script, the data dictionary fields, any optional fonts
and the file storage area are all stored in "external"
memory. These areas of external memory are allocated
25 numbers as used in the getbase(function,

A type MEMPTR is used to address this external memory. In
the include file custom.h the type MEMPTR must be defined.
If has less than 64k of memory allocated amongst the
30 external memory areas MEMPTR could be defined as unsigned
integer. If the memory is larger than this than MEMPTR
will normally be defined as "long".

Each block of external memory must APPEAR to be continuous.
35 That is incrementing a MEMPTR with the c ++ operator must
always generate a pointer to the next byte of the area.

The memory management routines must map these virtual memory addresses in to real memory addresses

5 getbase(base)

The function getbase returns the virtual memory address of each of the following blocks of memory

10 1 The smart card execution buffer

2The Script area - (includes the initialised data dictionary tables

15 3The Uninitialised data dictionary table area

4The file/ batch area

getDataByte

20

Returns the byte at the virtual address

uchar getDataInt(offset)

25 Returns the two byte value at the virtual address specified. The format of the two byte value is always Low/High regardless of the byte ordering of the microprocessor.

30 getScriptData

void getScriptData(uchar *bufferMEMPTR offset,int size)

Transfers data from the buffer to the virtual address

35 "offset"

setScriptData

void setScripfflata(uchar *buffer, OFFSETTYPE offset, int
5 size)

Either set external memory to NULL bytes or to a copy of a
buffer buffer is the memory buffer in the standard memory
area OR if NULL then the operation is like a memset

10 add notes on offset type

size is the number of bytes to store

For customising the CardScript command set
15

Adding Function Primitives

All function primitives have up to four parameters. Each
Parameter is of either one or two bytes length.

20 Numeric value parameters of values 0.. 1 27 are one byte in
length. Numeric values of greater length and in the
general format, with a maximum value of 4999.

25 All other parameters are in the general format, sixteen
bits High order first
bit 15 set - numeric value all other 15 bits contain the
value high nibble = 0x5 next three nibbles give string
number.

30 all other values high byte = table, low byte = field.

Reference

Index

Glossary

35 #

- 121 -

"start bomb()": <start-bomb>

"time-tick()": in the script-driver for processing 1/10
second time ticks